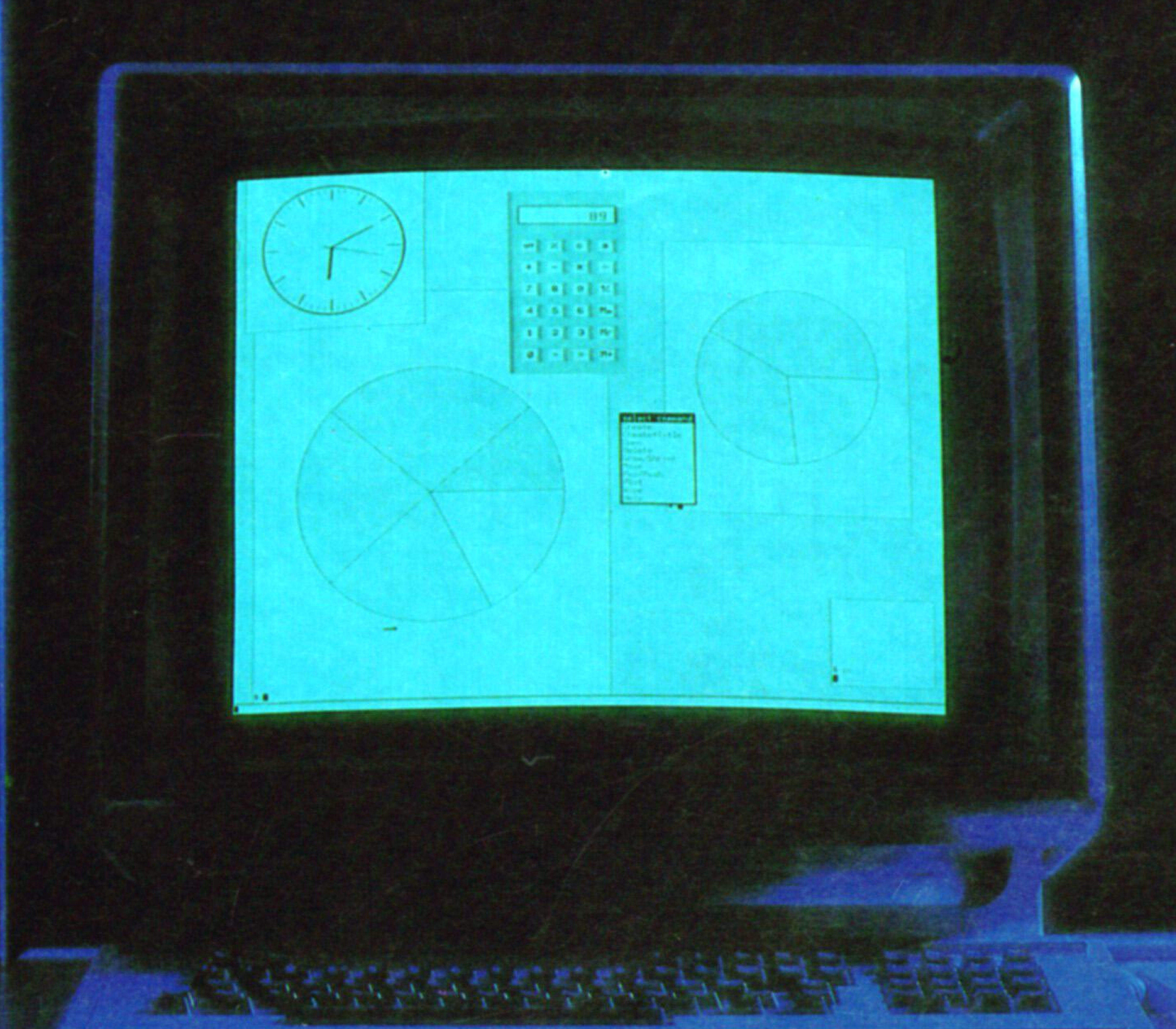
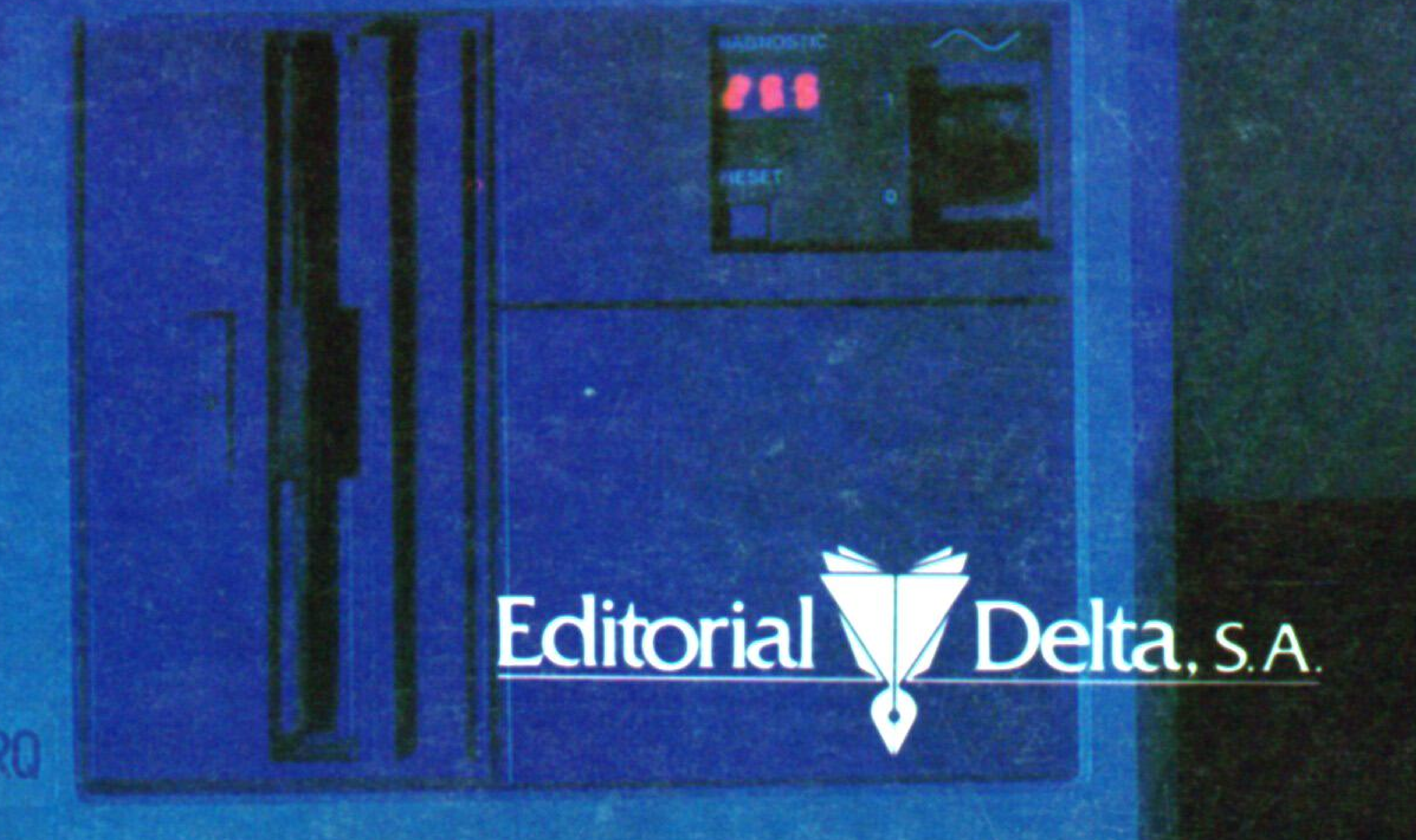
195 PTAS.
(IVA Incluido)



CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR





P.V.P. Canarias, Ceuta y Melilla 185 Ptas

Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen X-Fascículo 109

Director:	José Mas Godayol
Director editorial:	Gerardo Romero
Jefe de redacción:	Pablo Parra
Coordinación editorial:	Jaime Mardones Francisco Martín
Asesor técnico:	Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 - Télex: 93392 EPPA

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 120 fascículos de aparición semanal, encuadernables en diez volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-183-6 (tomo 10)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 188602
Impreso en España-Printed in Spain-Febrero 1986

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (120 fascículos más las tapas, guardas y transferibles para la confección de los 10 volúmenes) son las siguientes:

a) Un pago único anticipado de 27 105 ptas. o bien 10 pagos trimestrales anticipados y consecutivos de 2 711 ptas. (sin gastos de envío).

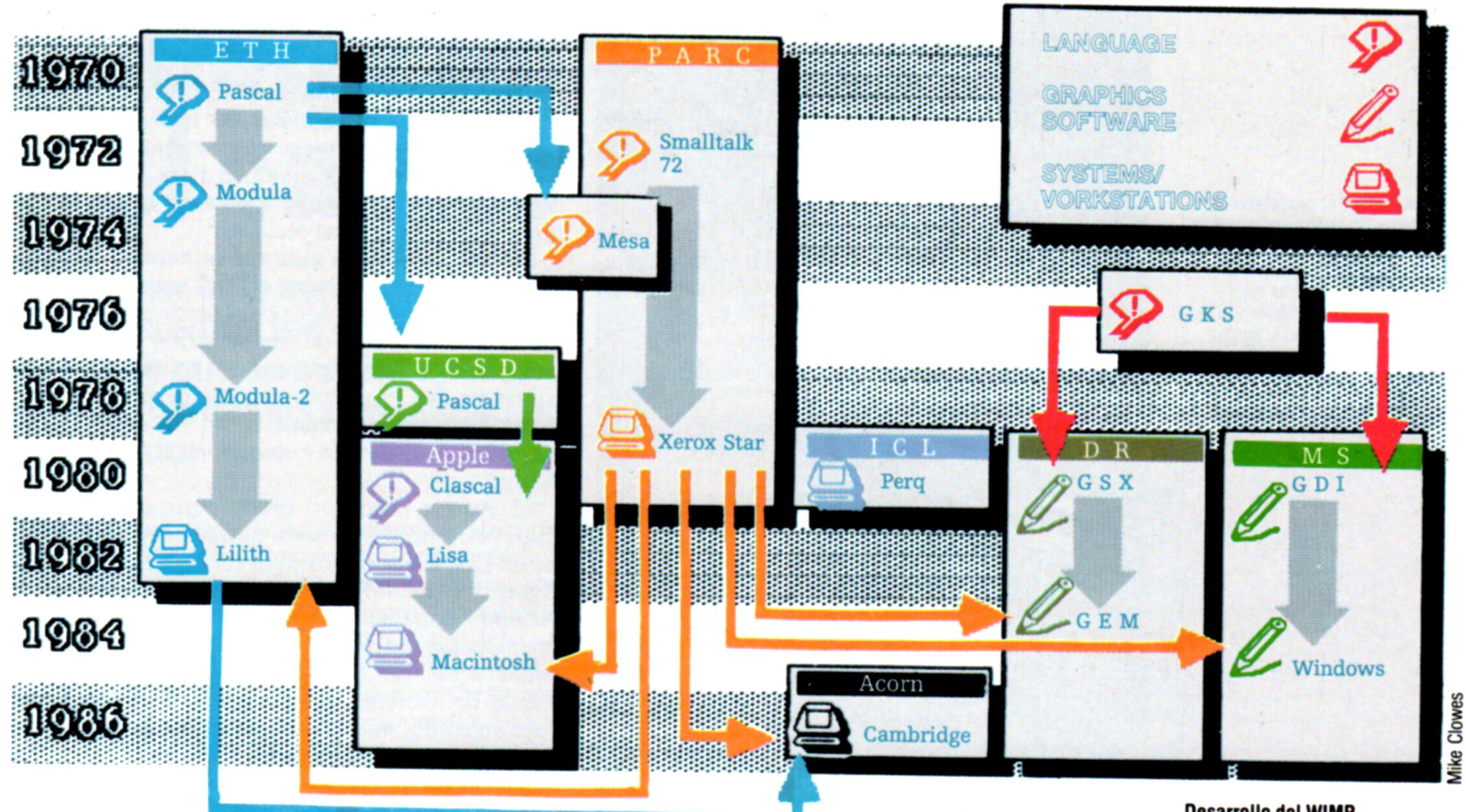
b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.





Desarrollo del WIMP
Las investigaciones realizadas en el PARC Xerox tuvieron influencia en muchas áreas diferentes del desarrollo de lenguajes y de hardware. Niklaus Wirth estuvo un año en el PARC y los resultados se pueden apreciar claramente en este caso en su posterior desarrollo del PASCAL y el MODULA-2. Muchos de los productos primeros están todavía disponibles. Por ejemplo, GKS (Graphics Kernel System), que proporciona un sistema de programación de gráficos independientes de la máquina, se ha convertido en una aplicación compleja (y onerosa), aunque continúa limitada a los usuarios y la investigación CAD-CAM. En el otro extremo de la escala, el GSX de Digital Research, una ampliación del sistema operativo, ha aparecido en el procesador de textos Amstrad PCW 8256, empaquetado con CP/M. Las abreviaturas utilizadas son las siguientes:

ETH: Elektro Technische Hochschule (profesor Niklaus Wirth)
PARC: Centro de investigación de Xerox en Palo Alto
UCSD: Universidad de California en San Diego
GKS: Graphics Kernel System (estandarizado en 1977)
DR: Digital Research Ltd
GSX: Graphics System Extension (DR)
GEM: Graphics Environment Manager (DR)
MS: Microsoft Corporation
GDI: Graphics Device Interface (MS)

Todo sobre los "blitters"

Proseguimos nuestro análisis de los antecedentes del desarrollo del GEM (Graphics Environment Manager) de Digital Research

El SMALLTALK y sus sucesores GEM, Macintosh y MS-Windows suponían grandes necesidades de hardware y, como vimos en el capítulo anterior, fueron los investigadores del PARC de Xerox quienes asumieron la tarea de desarrollar máquinas que fueran capaces de ejecutar los nuevos sistemas.

El Xerox Star se lanzó en 1980 y se servía en gran medida de la investigación de Xerox en el campo de la MMI (*man-machine interface*: interface hombre-máquina) y en el entorno SMALLTALK orientado hacia el objeto. El Star utilizaba un ratón de tres botones, con ventanas e iconos en una pantalla de mapa de bits en alta resolución. Este enfoque radical a la informática, junto con la poca disponibilidad de software y el costo de la máquina, determinaron que no fuera un éxito comercial.

No obstante, sí sirvió para lograr que la gente tomara conciencia de las posibilidades de los WIMP. Por ejemplo, en Gran Bretaña, ICL produjo la estación de trabajo Perq, tomando prestadas muchas de las características del Star. El Perq estaba dirigido al mundo científico y académico, y hacía hincapié en puntos tales como sus gráficos CAD y la capacidad de mezclar texto con gráficos. El sistema incluía una pantalla monocromática de alta definición, ventanas, menú y punteros de barras, y costaba alrededor de £10 000. Tuvo un moderado éxito comercial, vendiéndose en el mercado británico más de 2 500 unidades, si bien ello se debió en gran parte a su compatibilidad para la conexión en red con ordenadores centrales ICL. El Perq no fue tan radical como el Star, ¡de allí el éxito obtenido!

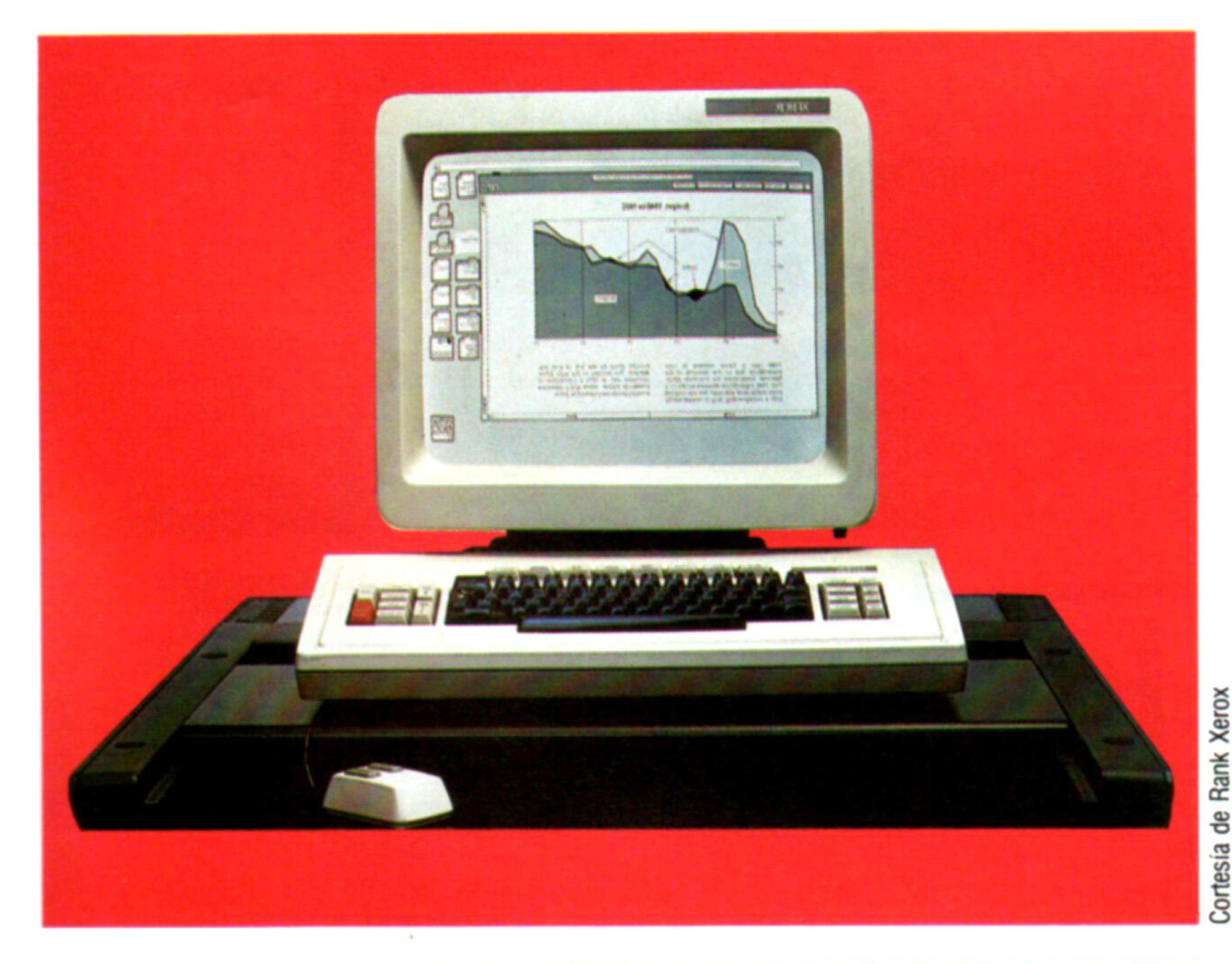
No fue hasta 1983, cuando se lanzó el Apple Lisa en el mercado mundial, que la tecnología WIMP atrajo realmente la atención de la industria informática. Comercializada con gran dinamismo y a un precio inferior a los 10 000 dólares (1 600 000 pts, aproximadamente), la máquina despertó un enorme interés. Sin embargo, se la criticó por su lento acceso a disco y su falta de software, y comercialmente se vio deslucida por el lanzamiento del IBM PC. Aunque rebautizada como Macintosh XL y experimentando una reducción de precio, su producción se suspendió en 1985.

Para implementar gráficos eficaces para iconos, menús y ventanas, es esencial una visualización por mapa de bits de gran velocidad. En el corazón del GEM, o de cualquier *kernel* de gráficos similar, debe haber un método muy potente para mover las imágenes a través de la pantalla. Aunque SMALLTALK era un sistema interpretado (no compilado),

La estrella del espectáculo
El Xerox Star ofrece numerosas facilidades desarrolladas a partir de los conceptos SMALLTALK originales. El sistema ofrece facilidades para gráficos y texto de fuentes múltiples y puntos múltiples, todo ello operando bajo un entorno WIMP

los algoritmos gráficos debían ser sumamente eficaces. Incluso sin considerar ninguna aplicación basada en gráficos, la tarea de gestionar las ventanas superpuestas y mantener actualizados sus cambiantes contenidos era una labor formidable. En este punto, una lentitud de respuesta incidiría en la "amabilidad" del sistema en su totalidad.

Además, las exigencias de ejecutar varios programas o tareas de forma concurrente (si bien simuladas mediante división de tiempo en una única CPU) hacían que el tiempo dedicado a manipular la pantalla debiera reducirse a la mínima expresión. Los potentes procesadores modernos de 16/32 bits, como el Motorola 68000 (utilizado en el Macintosh, Atari y Amiga) pueden alcanzar velocidades de alrededor de 2 mips (millones de instrucciones por segundo), pero la cantidad de código necesario para cuidar de la pantalla impone grandes demandas aun en estos procesadores. Para satisfacer las demandas de los sistemas operativos se han tenido que desarrollar nuevas técnicas.



Cortesía de Rank Xerox

La única forma de alcanzar un rendimiento eficaz es utilizar algoritmos "tejidos" en el firmware o chips especiales diseñados para llevar a cabo la manipulación adecuada de bits al recibir una instrucción. La CPU principal queda, entonces, libre para continuar con el cálculo y no es necesario escatimarle tiempo para manejar la VDU. Quizá el mejor ejemplo de este enfoque sea el Commodore Amiga.

El Amiga posee un 68000 equipado con tres chips periféricos exclusivos que se ocupan de canales de E/S. Estos dispositivos (denominados Portia, Agnes y Daphne) son responsables de gran parte del impresionante rendimiento de la máquina. Una visualización de alta resolución, por ejemplo, se puede actualizar muchas veces por segundo (con sonido de acompañamiento) para producir un dibujo animado, mientras se utiliza apenas el 9% del tiempo del procesador principal.

Si bien las modernas técnicas de diseño de chips han contribuido enormemente a la facilidad con que se pueden implementar tales chips para periféricos, el diseño de los algoritmos propiamente dichos también tuvo sus orígenes en los primeros trabajos realizados por Xerox. Al algoritmo SMALLTALK se lo llamó BITBLT (BIT BL*ock* T*ransfer*: transferencia de bloques de bits). En ocasiones se suele aludir al mismo coloquialmente como *bit-blit* o *blitter*. Al manipular imágenes, el BITBLT no las visualiza una y otra vez en la pantalla utilizando algoritmos para alterar la memoria de pantalla. En cambio, opera sobre la memoria de pantalla de forma directa e independientemente del procesador principal, explorando las posiciones de memoria de forma muy similar a la de un haz de electrones al barrer la pantalla de una VDU.

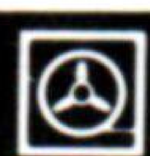
El BITBLT consigue su gran velocidad no sólo por explorar la memoria de este modo, sino también por el hecho de manipular la memoria de pantalla a nivel de bytes en lugar de bit a bit. Esto podría parecer una limitación seria; por ejemplo, ¿cómo alteramos los pixels de mapa de bits individuales si sólo podemos manipularlos en grupos de ocho? Y cuando deseamos desplazar una imagen a través de un fondo estático, ¿cómo restablecemos la imagen sobre la que previamente se ha escrito?

Los *blitters* resuelven este problema de un solo golpe utilizando operaciones lógicas binarias de gran velocidad: operando mediante AND, OR y XOR los bytes de la memoria de pantalla. El algoritmo puede generar una serie de máscaras de bytes que se superponen a la visualización existente. Estas operaciones lógicas son sumamente veloces y XOR, en particular, posee la ventaja de ser capaz de restaurar un fondo previamente sobreescrito.

La combinación de la tecnología de transferencia de bloques de bits, veloces procesadores de 16/32 bits como el Motorola 68000, y el moderno diseño de chips para dispositivos periféricos ha allanado el camino para los sistemas WIMP asequibles y eficaces.

Características de gráficos
El Perq, fabricado por ICL, fue el primer ordenador producido en Gran Bretaña que aplicó los conceptos pioneros de Xerox. La pantalla de alta resolución ofrecía un sobresaliente potencial de gráficos monocromáticos, y la máquina se vendió básicamente a establecimientos de investigación científica. Uno de los primeros usuarios fue la unidad de inteligencia artificial de la Universidad de Edimburgo, que utilizó las facilidades gráficas del Perq para (entre otras tareas) experimentos en el campo de la visión del robot

Marcus Wilson-Smith



Apostar en el casino

Al finalizar esta serie examinaremos los diversos sistemas que se suelen aplicar para aumentar las posibilidades del jugador



La fiebre del oro
Los casinos atraen a muchos tipos de jugadores diferentes, pero los apostadores serios constituyen una minoría. A pesar del margen de beneficios de que disfruta la casa en todos los establecimientos de apuestas, se han desarrollado algunos sistemas de predicción para maximizar las ganancias del apostador. Lamentablemente, la mayoría de los sistemas requieren un patrón de apuestas que el personal del casino puede reconocer con facilidad, y un apostador sistemático podría encontrarse con que se le invitara amablemente a abandonar la mesa

Imagebank

Cuando alguien le habla de un sistema de apuestas infalible, es comprensible que usted se muestre escéptico. Sin embargo, existen unos pocos sistemas que funcionan bien en las condiciones para las cuales fueron diseñados. Uno de tales sistemas lo inventó el matemático y filósofo francés Jean Le Rond d'Alembert (1717-1783), que colaboró con Diderot en la redacción de la *Enciclopedia*. Su sistema exige que las posibilidades reales a favor de usted sean del 50 % o superiores. En general esto no es así, pero si el apostador posee un buen programa de predicción puede resultar valedero. El propio D'Alembert utilizó su sistema de apuestas para amasar una considerable suma en los días en que las ruedas de la ruleta eran imparciales y no había ventaja para la casa.

En la actualidad, la ruleta es peor que ineficaz (a menos que usted encuentre una bola que se incline a su favor), pero si su programa de predicción está dando algo más que un reembolso a la par, este método puede serle rentable. Funciona así:

- Empezar con una apuesta de 5 unidades.
- Tras cada apuesta ganada reducir la apuesta en una unidad.
- Tras cada apuesta perdida aumentar la apuesta en una unidad.
- Si llega a quedar en cero, recomenzar con 5 unidades.

La eficacia del método es muy compleja, pero se puede demostrar empíricamente con un ordenador. Nosotros hemos escrito un programa de demostración para darle una idea de lo que cabe esperar de tal sistema. Esto es bastante fácil de hacer, porque el BASIC posee un generador de números aleatorios incorporado (la función RND).

Con este programa usted puede variar la probabilidad de ganar una apuesta, y usarla para ver cómo incide ello en el rédito. Asimismo, puede adaptarlo para probar otras estrategias de apuestas ideadas por usted mismo, sin perder ni un céntimo. De hecho, ¡la gran ventaja de las simulaciones como ésta es que lo mantienen a uno al margen del local de apuestas! Proporcionan el escenario para probar sus teorías y descartarlas sin que su bolsillo corra ningún riesgo. Y ocasionalmente también podrían confirmar que usted ha descubierto un plan que vale la pena.

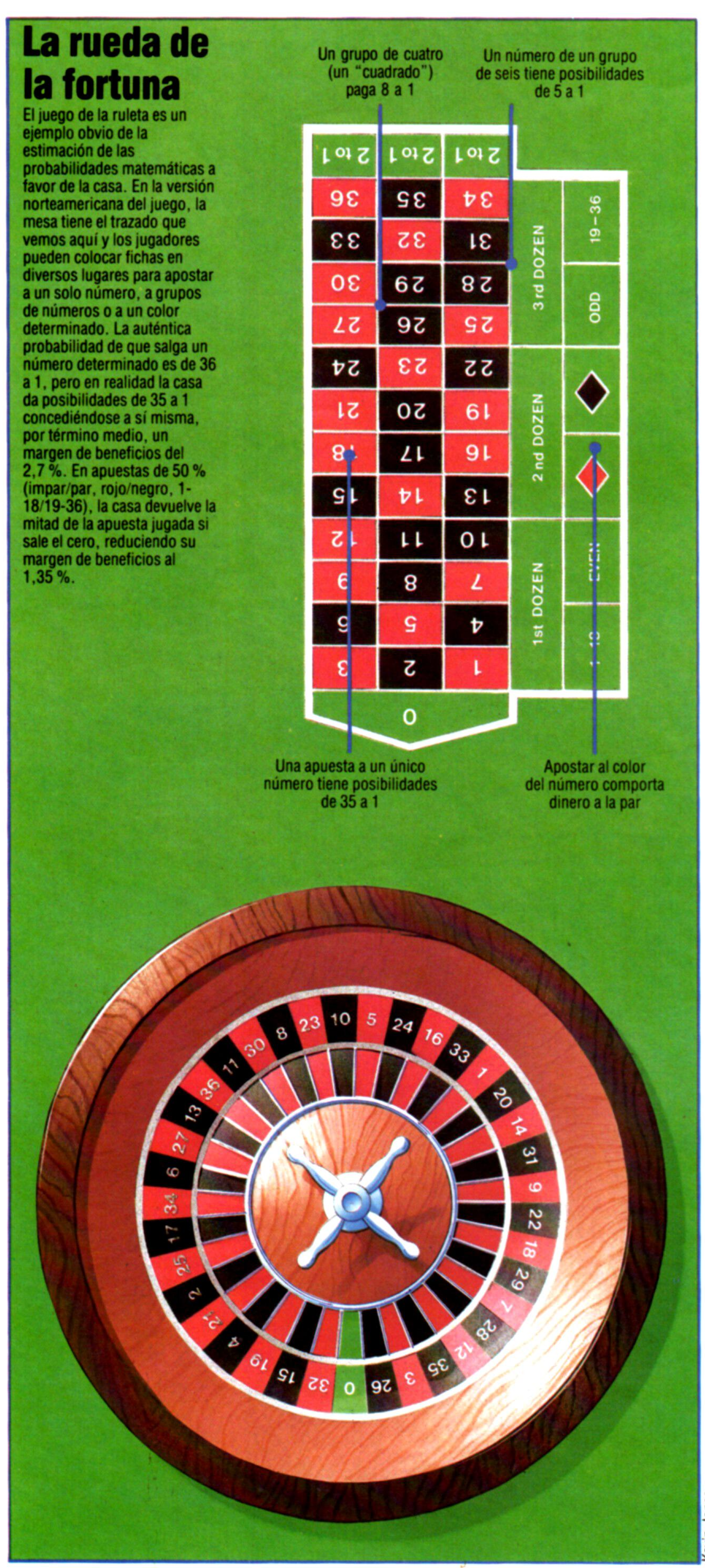
La esencia del método de D'Alembert es que magnifica las expectativas de ganancias. Sin embargo, si las posibilidades están en su contra, también magnifica sus pérdidas. En otro juego de casino (el *blackjack*) se puede utilizar una estrategia diferente para revertir la ventaja de la casa a su favor. Usted simplemente debe apostar fuerte en las manos buenas y apostar el mínimo permitido en las malas. Como es natural, para poder hacerlo tiene que distinguir entre manos buenas y malas, convirtiéndose en un *calculador* o *contador*. Es decir, debe memorizar los naipes que ya se han jugado.

Todos los sistemas contadores del *blackjack* provienen del que utilizara Edward Thorp a principios de los años sesenta. Este hombre ganó una fortuna en Las Vegas y escribió un libro sobre sus métodos, cuyo título es *Beat the dealer*. Si bien el empleo cuidadoso de su sistema inclina las posibilidades a su favor, posee dos inconvenientes principales:

1. El coeficiente esperado de rédito sobre el movimiento total es del orden del 0,5 %.
2. El personal de los casinos está entrenado para detectar a los contadores.

El primer problema proviene del segundo. Dado que el sistema se puede detectar fácilmente, las versiones sofisticadas del sistema exigen un comportamiento más sutil, lo que limita las ganancias potenciales. Esto significa que si apuesta 1 000 pts. en una hora sólo puede esperar ganar unas 40 pts.

El segundo problema significa que, tras manifestar durante cierto tiempo el perfil de apuestas de un contador eficiente, es probable que el gerente del casino lo invite a una cena por cuenta de la casa. Si usted se niega, es casi seguro que un robusto empleado del club lo "escolte" hacia la salida del local. En consecuencia, abandonemos la atmósfera saturada de humo del casino e introduzcámonos en un área que hasta ahora, y sorprendentemente, no ha sido tenida en cuenta: la aplicación de las técnicas de inteligencia artificial (AI) en las apuestas.



La rueda de la fortuna

El juego de la ruleta es un ejemplo obvio de la estimación de las probabilidades matemáticas a favor de la casa. En la versión norteamericana del juego, la mesa tiene el trazado que vemos aquí y los jugadores pueden colocar fichas en diversos lugares para apostar a un solo número, a grupos de números o a un color determinado. La auténtica probabilidad de que salga un número determinado es de 36 a 1, pero en realidad la casa da posibilidades de 35 a 1 concediéndose a sí misma, por término medio, un margen de beneficios del 2,7 %. En apuestas de 50 % (impar/par, rojo/negro, 1-18/19-36), la casa devuelve la mitad de la apuesta jugada si sale el cero, reduciendo su margen de beneficios al 1,35 %.

Kevin Jones

Uno de los desarrollos más interesantes a que ha dado lugar la AI en la última década es el sistema experto. Como hemos visto en series anteriores, un sistema experto comparte muchas de las características de un experto humano, incluyendo la capacidad de manejar inferencias inciertas en situaciones mal definidas mediante la lógica formal o el razonamiento basado en la probabilidad.

Cuando se está desarrollando un sistema experto para usar en prospecciones de minerales preciosos o diagnosticar enfermedades, lo primero que hacen los diseñadores es procurarse la ayuda de un especialista humano en la materia que pueda realizar la tarea con un elevado nivel de destreza. Luego se codifica el conocimiento del experto y se lo refina para que lo utilice el ordenador. El mismo enfoque se puede utilizar, por ejemplo, con las carreras de caballos.

Un investigador de la Carnegie-Mellon University de Pittsburgh, Stephen F. Smith, ha desarrollado un programa de aprendizaje llamado LS-1 que aprendió a jugar al póquer cerrado.

El póquer cerrado es un juego para dos o más jugadores. A cada jugador se le reparten cinco naipes. Los jugadores, por turno, tienen la opción de pedir a los demás que muestren su mano, apostar o abandonar. Cuando se realiza la primera vuelta, cada jugador puede sustituir hasta tres naipes de su mano con naipes nuevos (no vistos) de la baraja. Un abandono da por terminada la actual ronda de juego. Una apuesta consiste en colocar algo de dinero, al menos tanto como la apuesta anterior, en un "bote" central. Si se realiza una segunda vuelta, se muestran todas las manos y el jugador que tenga la mano de mayor jerarquía se queda con todo el dinero del bote.

El LS-1 es un sistema de aprendizaje que emplea un algoritmo de adaptación evolutiva o *genética*. Su bucle principal opera del siguiente modo:

1. Generar al azar unas reglas de partida.
2. Evaluar todas las reglas y si la puntuación media es suficientemente alta, parar y visualizarlas.
3. De lo contrario, calcular para cada regla su probabilidad de selección $p=e/E$, donde e es su puntuación individual y E la puntuación global de todas las reglas.
4. Generar la siguiente población mediante una selección basada en las probabilidades calculadas en 3 y aplicando ciertos operadores genéticos; luego repetir desde el paso 2.

Cada pasada por este bucle corresponde a una única generación.

Los operadores genéticos mencionados anteriormente son cruce, inversión y mutación. Intentan simular, de forma simplificada, lo que sucede en el proceso de la evolución. El cruce es un procedimiento de apareamiento en virtud del cual se combina la información de dos estructuras paternas para conformar un nuevo *descendiente*, es decir, una regla candidata a la comprobación. La inversión reordena la información de la regla. La mutación introduce cambios caprichosos.

Es evidente que esta competencia entre estructuras de reglas tiene mucho en común con la competencia entre organismos naturales. Las reglas que tienen las mejores posibilidades de dejar descendientes son aquellas que se desempeñan mejor en la tarea.



En el caso del póquer cerrado, los resultados fueron impresionantes. Se enfrentó al LS-1 con un programa de póquer hecho a mano, puesto que ningún oponente hubiera tenido paciencia suficiente para jugar más de 40 000 rondas. Para cada decisión de apuesta tuvo cuatro acciones alternativas: apuesta fuerte, apuesta baja, verlas o abandono. Comenzando a partir de nada, con un conjunto de reglas iniciales aleatorias, LS-1 demostró ser tan fuerte que hubo que reprogramar al ordenador oponente para convertirlo en un rival más preparado. El programa modificado resultó una prueba más dura, pero al cabo de varios miles de rondas el LS-1 le ganó nueve de cada diez manos. Cuando terminó, fue uno de los jugadores de póquer no humanos más fuertes del mundo.

Otro estudio piloto, también con regustos evolutivos, se llevó a cabo utilizando el sistema de aprendizaje BEAGLE (*Biological Evolutionary Algorithm Generating Logical Expressions*: algoritmo de evolución biológica generador de expresiones lógicas) sobre una muestra de datos de carreras de caballos celebradas en Gran Bretaña en 1982.

Los datos comprendían a 153 caballos a partir de 51 hándicaps de todas las edades con 10 corredores o menos. En el pronóstico de apuestas de cada carrera sólo se tuvieron en cuenta los tres mejores caballos. La base de datos se dividió en dos partes: se utilizaron 99 registros como conjunto de entrenamiento (para formar las reglas) y los 54 restantes se utilizaron como datos de prueba (para ver si las reglas se generalizaban en ejemplos nuevos). Cada caballo se midió sobre 18 variables, incluyendo:

VELOCIDAD	cifra de velocidad para el caballo
ESCALADO	Si la cifra de velocidad es la más alta, casi la más alta, etc.
FC	posición en el pronóstico
ÚLTIMA	lugar en última salida
PENÚLT.	lugar en penúltima salida
ANTEPE-NÚLTIMA	lugar en antepenúltima salida
DIAS	días desde la última carrera
SPOTFORM	evaluación según el *Daily Mirror*
BIN	mejor promedio de dist. reciente
GOING	estado de la pista
WT	peso del jockey

Además, se usó la variable GANA para registrar si el caballo ganó.

El programa BEAGLE utiliza un esquema de aprendizaje evolutivo similar al que describimos en la serie dedicada a la inteligencia artificial (p. 1802). Al proporcionársele los datos de las carreras de caballos, salió con reglas tales como (VELOCIDAD>60) y (PESO> (VELOCIDAD * 2.18)), que entre ambas ayudan a distinguir entre probables ganadores y perdedores. Estas reglas se utilizan conjuntamente para dar una "rúbrica" o "huella digital". Es decir, si las reglas 1 y 3 son verdaderas y las reglas 2 y 4 falsas, la rúbrica sería el número binario 0101 (porque el sistema trabaja hacia atrás) o 5 decimal. Este índice 5 señala la posición 5 de una tabla en la cual se acumula información sobre esa combinación de valores de reglas.

Cuando el módulo LEAF (*Logical Evaluator And Forecaster*: evaluador y pronosticador lógico) aplicó las reglas generadas por ordenador a los datos no vistos, las mismas resultaron correctas el 80 % de las veces. Por supuesto, usted puede acertar el 72 % de las veces, simplemente diciendo "no", porque la mayoría de los caballos pierden. Pero un examen de los resultados impresos demuestra que las reglas actuaron como un sistema de filtro muy eficaz. El BEAGLE sólo pronosticó cuatro ganadores: todos ellos ganaron. Ésta es una buena señal, porque la selectividad es la esencia de la apuesta científica. (Uno no puede esperar pronosticar todas las carreras; lo importante es esperar hasta hallar "un caballo de carreras entre burros" y entonces dar el golpe.) Además, sus 13 mejores pronósticos incluyeron 10 ganadores y sólo tres perdedores. Dicho en otras palabras, los nueve caballos del grupo de probabilidad 0,4854 lo hicieron mejor de lo que se esperaba, con seis ganadores.

Analizando esto desde otro ángulo, la cantidad de ganadores entre los 41 peores fue de sólo cinco. La categoría inferior, los "sin esperanzas", contuvo sólo un error, de modo que es eficiente para filtrar los perdedores. (Observe que el LEAF coloca signos de interrogación en el listado de pronósticos para señalar las predicciones efectuadas sobre la base de muestras pequeñas.)

En líneas generales, entonces, existe la evidencia de que las técnicas de aprendizaje desarrolladas por la investigación en AI pueden ser rentables en el hipódromo. Esto es apenas un estudio preliminar, pero los resultados son muy alentadores. En la literatura dedicada al estudio de la AI hay gran riqueza de técnicas a la espera de ser explotadas.

Simulador de apuestas

```
10 REM ******************************
15 REM ** METODO DE D'ALEMBERT: ******
20 REM ******************************
75 @ %=4
100 PRINT'"Simulador de apuestas:"
101 REPEAT
110   PRINT
120   PRINT "Por favor dar probabilidades de exito ";
122   INPUT PS
125   UNTIL PS > 0 AND PS < =1
130 INPUT "Cuantos ensayos ",T
140 N=0
150 GANANCIAS=0: PERDIDAS =0
155 ULTIMA=0:ST=-99
160 W=0
190 REM -- Bucle principal:
200 REPEAT
202   N=N+1
210   GOSUB 1000: REM calcular apuesta
212   PRINT N;" ";
220   IF RND(1) < = PS THEN S=1 ELSE S=0
230   IF S THEN GOSUB 1500 ELSE GOSUB 1600
240   ULTIMA=S
250   UNTIL N> = T
300 PRINT: PRINT "Total ganado = "; GANANCIAS-PERDIDAS
330 PRINT "Proporcion de exitos: ";W/T
360 END
999 :
1000 REM -- Rutina decision apuesta:
1010 REM se puede alterar para experimentar:
1020 IF ULTIMA THEN ST=ST-1 ELSE ST=ST+1
1030 IF ST < 1 THEN ST=5
1040 RETURN
1050 :
1500 REM -- Rutina exito:
1510 GANANCIAS=GANANCIAS+ST
1520 PRINT "Gano ";ST;TAB(15);"Ganancias= ";GANANCIAS-PERDIDAS
1525 W=W+1
1550 RETURN
1560 :
1600 REM -- Rutina fracaso:
1610 PERDIDAS=PERDIDAS+ST
1620 PRINT "Perdido ";ST;TAB(15);"Ganancias = ";GANANCIAS-PERDIDAS
1650 RETURN
1660 :
```

Hoja completa

Finalmente, crearemos rutinas que nos permitan guardar hojas completas de datos y fórmulas

La última adición a nuestro programa de hoja electrónica es la implementación de las rutinas para guardar y cargar, que nos permiten crear registros permanentes de datos o fórmulas en disco o cinta. Puesto que cada una de las cuatro máquinas para las que está diseñada la hoja electrónica trata los archivos de forma diferente, incluimos listados separados para la gama Amstrad, el Commodore 64, el BBC Micro y el Sinclair Spectrum.

En las cuatro versiones las rutinas cargar y guardar empiezan en la línea 7000, visualizando un corto submenú para las diversas opciones disponibles. Éstas son:

```
1: Cargar una hoja de DATOS
2: Cargar una matriz de FORMULAS
3: Guardar una hoja de DATOS
4: Cargar una hoja de FORMULAS
"X": Para salir
```

A partir del menú de opciones, usted puede ver que el programa le permite guardar o cargar independientemente datos o fórmulas. Esta característica es especialmente útil porque le permite crear una plantilla de fórmulas para una tarea determinada (supongamos, calcular las devoluciones del IVA), guardarla en cinta o disco o luego cargarla cada vez que la necesite, para poder trabajar cada vez con un nuevo conjunto de datos. Por el contrario, quizá tenga una tarea que requiera realizar varias operaciones diferentes sobre el mismo conjunto de datos, en cuyo caso el conjunto original de datos se puede almacenar y volver a cargar cuando así se necesite para cada nuevo cálculo.

Implementar esta facilidad para cargar y guardar fórmulas o datos individuales en realidad es muy fácil. Usted recordará que todos los valores de datos para la hoja están retenidos en la matriz numérica bidimensional M(,), y que las correspondientes fórmulas de celdas están retenidas en una matriz alfanumérica unidimensional, F$(). Todo cuanto habremos de hacer es guardar la matriz adecuada como un archivo secuencial. Para volver a cargarla, sólo debemos leer nuevamente el archivo en la matriz de la cual se tomó, perdiendo, como es lógico, el contenido que la matriz tenga en ese momento.

Comencemos nuestro análisis de cómo guarda el programa las matrices examinando las versiones para el Amstrad CPC y el BBC Micro. Estas máquinas emplean una sintaxis similar para abrir y cerrar archivos. OPENIN abre un archivo para una entrada y OPENOUT abre un archivo sobre el cual se ha de escribir, pero mientras que el sistema de tratamiento de archivos del BBC Micro destina un número de canal como parte de la operación de apertura del archivo, el Amstrad sólo permite utilizar un único canal (# 9) para sus operaciones PRINT e INPUT sobre archivos.

Las rutinas para guardar simplemente recorren la matriz, elemento a elemento, utilizando un bucle FOR...NEXT (o, en el caso de la matriz bidimensional M(,), un par de bucles anidados) para grabarlos en el archivo secuencial. La rutina para cargar no es más que el mismo proceso pero en sentido inverso.

Si bien el Spectrum no soporta archivos secuenciales, le permitirá simular el proceso guardando matrices numéricas o alfanuméricas en cinta empleando la sintaxis:

```
SAVE I$ DATA F$()
```

donde I$ es el nombre del archivo y F$(), la matriz a guardar.

Dado que todo cuanto necesitamos hacer es guardar matrices, este sistema resulta ideal para nuestras aplicaciones.

La versión Commodore

Por último, llegamos a la versión para el Commodore 64. Al desarrollar el programa, quizá le parezca que hay un error en el BASIC Commodore. Un archivo secuencial se puede preparar a partir de una matriz numérica y volver a leerlo sin ninguna dificultad.

El problema surge al escribir los elementos de una matriz alfanumérica en un archivo.

Cuando se DIMensiona una matriz alfanumérica, todos los elementos se establecen inicialmente en series nulas, representadas en el sistema Commodore mediante CHR$(0). Si imaginamos que utilizamos la hoja electrónica para programar algunas fórmulas, pero dejando algunas celdas sin fórmulas, los elementos correspondientes de la matriz de fórmulas continuarán nulos y cuando se guarde F$() se escribirán en el archivo como CHR$(0). El problema se plantea cuando se vuelve a leer el archivo mediante INPUT #.

Si bien esta instrucción leerá cualquier fórmula que se haya establecido antes de guardarla, no volverá a leer aquellos elementos de la matriz que fueran nulos y el sistema quedará colgado. Usted puede sortear esta "característica" comprobando cada elemento de la matriz alfanumérica antes de escribirlo en el archivo, y si el elemento es cero, puede escribir CHR$(1) en el lugar del CHR$(0) que normalmente el Commodore 64 escribiría de forma automática. Aunque CHR$(1) no representa ningún carácter, sortea el fracaso del INPUT # en reconocer CHR$(0) y, por lo tanto, todo lo que hemos de hacer es modificar la rutina que vuelve a leer el archivo en F$(). Esto significa que los elementos leídos como CHR$(1) se establecen como la serie nula (ver líneas 7425 y 7230 de la versión para el Commodore).

Las rutinas que aparecen en el listado para el Commodore 64 son para sistemas de disco. Para



sistemas de cinta, introduzca los siguientes cambios:

```
7110 GOSUB 7500:OPEN 1,1,0,I$
7219 GOSUB 7500:OPEN 1,1,0,I$
7310 GOSUB 7500:OPEN 1,1,1,I$
7410 GOSUB 7500:OPEN 1,1,1,I$
```

Y así llegamos al final de nuestra serie sobre hojas electrónicas. Al comenzarla habíamos esbozado algunas de las limitaciones de nuestro programa. Por ejemplo, el ancho de la celda se fija en cinco caracteres, en una celda sólo se pueden entrar datos numéricos y se ha simplificado la detección de errores en las entradas para acortar el listado. Como un proyecto para el futuro, quizá le interese añadir y ampliar algunas de las características que hemos presentado.

HELP!

La subrutina de la línea 6000 proporciona una pantalla de ayuda que indica las teclas que se han de pulsar para obtener las diversas funciones. Lo que no le dice es para qué sirve cada función, de modo que nos referiremos a cada una de las funciones de la hoja describiendo su funcionamiento.

ENTRAR FORMULA NUEVA EN CELDA: Se puede entrar en la celda de la posición actual del cursor. Las fórmulas realizan operaciones aritméticas con constantes y valores de celdas (a las que se alude por nombre, como A1, B12) y se utilizan para dirigir la función CALCULAR de la hoja electrónica.

ALMACENAR HOJA ACTUAL: Los datos se pueden almacenar temporalmente dentro del programa. Esta facilidad se utiliza para poner a prueba estrategias de cálculo diferentes en la misma hoja de datos, o usar como mecanismo de seguridad para que los datos no se corrompan a consecuencia de una dirección errónea de la función CALCULAR.

TOMAR HOJA ALMACENADA: Recupera una hoja de datos almacenada previamente.

CALCULAR: Utiliza las fórmulas y los datos entrados en la hoja para dirigir sus operaciones, colocando los resultados de los cálculos en celdas adecuadas.

REPRODUCIR: Es útil poder reproducir una fórmula de una celda determinada a través de una fila o a lo largo de una columna. La sintaxis requerida es N1 (N2-N3), donde N1 es el nombre de la celda cuya fórmula se ha de reproducir, y N2 y N3 son los límites entre los cuales ha de tener lugar la reproducción. La rutina selecciona automáticamente la reproducción en fila o en columna.

BORRAR HOJA: Restablece a cero los datos de todas las celdas.

TAB CURSOR: El cursor de la hoja electrónica se puede desplazar mediante el uso de las teclas del cursor, pero en virtud de esta función se pueden efectuar saltos a una celda determinada. Entre el nombre de la celda a la cual usted desea ir.

RUTINAS CARGAR/GUARDAR: Selecciona el submenú cargar/guardar, que a su vez le permite guardar los datos o fórmulas en curso en disco o cinta, y cargar los almacenados con anterioridad.

Pantallas HELP y tratamiento de archivos

Commodore 64:

```
6000 REM ***** PANTALLA DE AYUDA *****
6010 PRINT CHR$(147):PRINT
6020 PRINT:PRINT "                F1 – IMPRIME ESTA
     PANTALLA HELP "
6030 PRINT:PRINT "                F2 – ENTRAR NUEVA
     FORMULA EN CELDA "
6040 PRINT:PRINT "                F3 – ALMACENAR
     HOJA ACTUAL "
6050 PRINT:PRINT "                F4 – TOMAR HOJA
     ALMACENADA "
6060 PRINT:PRINT "                F5 – CALCULAR
     HOJA "
6070 PRINT:PRINT "                F6 – BORRAR
     HOJA "
6080 PRINT:PRINT "                F7 – REPRODUCIR
     FORMULA "
6090 PRINT:PRINT "                F8 – CARGAR/GUARDAR
     HOJAS "
6100 PRINT:PRINT "                'G' – PARA
     TABULAR CURSOR
6110 GET A$:IF A$="" THEN 6110
6120 GOSUB 1000:GOSUB 1700:RETURN
7000 REM ***** RUTINAS CARGAR/GUARDAR *****
7010 PRINT CHR$(147):PRINT:PRINT
7020 PRINT:PRINT "                F1 – CARGAR HOJA
     DATOS "
7030 PRINT:PRINT "                F3 – CARGAR
     MATRIZ FORMULAS "
7040 PRINT:PRINT "                F5 – GUARDAR HOJA
     DATOS "
7050 PRINT:PRINT "                F7 – GUARDAR
     MATRIZ FORMULAS "
7055 PRINT:PRINT "                'X' – PARA SALIR "
7056 PRINT:PRINT:PRINT
7060 GET A$:IF A$="" THEN 7060
7070 IF A$=CHR$(133) THEN 7100
7080 IF A$=CHR$(134) THEN 7200
7090 IF A$=CHR$(135) THEN 7300
7095 IF A$=CHR$(136) THEN 7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB 1700:
     RETURN
7100 REM **** CARGAR DATOS ****
7110 GOSUB 7500:I$="0:"+I$+",S,R":OPEN
     2,8,2,I$
7120 FOR I=1 TO 15:FOR J=1 TO 15
7130 INPUT #2,M(I,J)
7140 NEXT J,I
7150 CLOSE2
7160 GOTO 7010
7200 REM **** CARGAR MATRIZ FORMULAS ****
7210 GOSUB 7500:I$="0:"+I$+       ":OPEN
     ,S,R":OPEN 2,8,2,I$
7220 FOR I=1 TO 255
7230 INPUT # 2,F$(I):IF F$(1)=CHR$(1) THEN
     F$(I)=""
7240 NEXT 1
7250 CLOSE2
7260 GOTO 7010
7300 REM **** GUARDAR DATOS ****
7310 GOSUB 7500:I$="0:"+I$+",S,W":OPEN 2,8,2,I$
7320 FOR I=1 TO 15:FOR J=1 TO 15
7330 PRINT # 2,M(I,J)
7340 NEXT J,I
7350 CLOSE2
7360 GOTO 7010
7400 REM **** GUARDAR MATRIZ FORMULAS ****
7410 GOSUB 7500:I$="0:"+I$+ ",S,W":OPEN
     2,8,2,I$
7420 FOR I=1 TO 255
7425 IF F$(I)="" THEN PRINT # 2,CHR$(1):
     GOTO 7440
7430 PRINT #2,F$(I)
7440 NEXT I
7450 CLOSE2
7460 GOTO 7010
7500 INPUT "ENTRAR NOMBRE ARCHIVO:";I$
7510 RETURN
```

F1: Imprime pantalla HELP
F2: Entrar nueva fórmula en celda
F3: Almacenar hoja actual
F4: Tomar hoja almacenada
F5: Calcular hoja
F6: Borrar hoja
F7: Reproducir fórmula
F8: Rutinas cargar/guardar
"G": Tabular cursor



BBC Micro:

"H": Pantalla HELP
"F": Entrar nueva fórmula en celda
"S": Almacenar hoja actual
"G": Tomar hoja almacenada
"C": Calcular hoja
"Z": Borrar hoja
"R": Reproducir fórmula
"T": Tabular cursor
"D" Rutinas cargar/guardar

```
1189 IF A$="D" THEN GOSUB 7000:
     REM RUTINAS CARGAR
     GUARDAR
6000 REM **** PANTALLA DE AYUDA ****
6020 CLS:PRINT TAB(6,2)"H – IMPRIMIR ESTA
     PANTALLA HELP"
6030 PRINT TAB (6,4)"F – ENTRAR NUEVA
     FORMULA EN CELDA"
6040 PRINT TAB (6,6)"S – ALMACENAR HOJA
     ACTUAL "
6050 PRINT TAB (6,8)"G – TOMAR HOJA
     ALMACENADA"
6060 PRINT TAB (6,10)"C – CALCULAR HOJA"
6070 PRINT TAB (6,12)"Z – BORRAR HOJA"
6080 PRINT TAB (6,14)"R – REPRODUCIR
     FORMULA"
6090 PRINT TAB (6,16)"T – TABULAR A NUEVA
     POS/ CURSOR"
6100 PRINT TAB (6,18)"D – RUTINAS
     CARGAR/GUARDAR"
6110 A$=GET$:GOSUB 1000:GOSUB
     1700:RETURN
7000 REM *** RUTINAS CARGAR /
     GUARDAR *****
7010 CLS:PRINT TAB (6,4)"1 – CARGAR HOJA
     DATOS"
7030 PRINT TAB (6,6)"2 – CARGAR MATRIZ
     FORMULAS"
7040 PRINT TAB (6,8)"3 – GUARDAR HOJA
     DATOS"
7050 PRINT TAB (6,10)"4 – GUARDAR MATRIZ
     FORMULAS"
7055 PRINT TAB (6,12)"'X' – PARA
     SALIR"
7056 A$=GET$:A=VAL(A$)
7060 IF A > 0 AND A < 5 THEN ON A GOTO
     7100,7200,7300,7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB
     1700:RETURN
7100 REM ***** CARGAR DATOS *****
7110 GOSUB 7500:F=OPENIN I$
7120 FOR I=1 TO 15:FOR J=1
     TO 15:INPUT # F,M(I,J):
     NEXT J,I
7130 CLOSE # F: GOTO 7010
7200 REM **** CARGAR FORMULAS ****
7210 GOSUB 7500:F=OPENIN I$
7220 FOR I=1 TO 255:INPUT # F,F$(I):
     NEXT I
7230 CLOSE # F:GOTO 7010
7300 REM **** GUARDAR DATOS ****
7310 GOSUB 7500:F=OPENOUT I$
7320 FOR I=1 TO 15:FOR J=1
     TO 15:PRINT # F,M(I,J):
     NEXT J,I
7330 CLOSE # F:GOTO 7010
7400 REM **** GUARDAR FORMULAS ****
7410 GOSUB 7500:F=OPENOUT
     I$
7420 FOR I=1 TO 255:INPUT # F,F$(I):
     NEXT I
7430 CLOSE # F:GOTO 7010
7500 INPUT TAB(0,22)"NOMBRE
     ARCHIVO"I$
7510 RETURN
```

Amstrad CPC 464/664:

"H": Pantalla HELP
"F": Entrar nueva fórmula en celda
"S": Almacenar hoja actual
"G": Tomar hoja almacenada
"C": Calcular hoja
"Z": Borrar hoja
"R": Reproducir fórmula
"T": Tabular cursor
"D": Rutinas cargar/guardar

Nota: Pulsar CAPS LOCK antes de la ejecución

```
1175 IF A$="Z" THEN GOSUB 5000:REM
     BORRAR LA HOJA
1177 IF A$="S" THEN GOSUB 5150:REM
     ALMACENAR HOJA
1178 IF A$="H" THEN GOSUB 6000:REM
     PANTALLA AYUDA
1189 IF A$="D" THEN GOSUB 7000:REM
     RUTINAS CARGAR/GUARDAR
6000 REM **** pantalla de ayuda ****
6020 CLS:LOCATE 6,3:PRINT"H – Imprimir esta
     pantalla HELP"
6030 PRINT:PRINT TAB(6)"F – Entrar nueva
     FORMULA"
6040 PRINT:PRINT TAB (6)"S – ALMACENAR
     hoja actual"
6050 PRINT:PRINT TAB(6)"V – Tomar hoja
     ANTERIOR"
6060 PRINT:PRINT TAB(6)"C – CALCULAR hoja"
6070 PRINT:PRINT TAB(6)"Z – BORRAR hoja"
6080 PRINT:PRINT TAB(6)"R = REPRODUCIR
     formula"
6090 PRINT:PRINT TAB(6)"G – IR a una nueva
     celda"
6100 PRINT:PRINT TAB(6)"D – Rutinas
     GUARDAR/CARGAR"
6110 WHILE INKEY$="":WEND
6120 GOSUB 1000:GOSUB 1700:RETURN
7000 REM *** RUTINAS CARGAR/GUARDAR ***
7010 CLS:LOCATE 6,4:PRINT"1 – CARGAR HOJA
     DATOS"
7020 PRINT:PRINT TAB(6)"2 – CARGAR MATRIZ
     FORMULAS"
7030 PRINT:PRINT TAB(6)"3 – GUARDAR
     MATRIZ DATOS".
7040 PRINT:PRINT TAB(6)"4 – GUARDAR
     MATRIZ FORMULAS"
7050 PRINT:PRINT TAB(6)"'X' – PARA SALIR"
7055 A$="":WHILE A$="":A$=INKEY$:WEND
7056 A=VAL(A$)
7060 ON A GOTO 7100,7200,7300,7400
7096 IF A$="X" THEN GOSUB 1000:GOSUB
     1700:RETURN
7100 REM **** CARGAR DATOS ****
7110 GOSUB 7500:OPENIN I$
7120 FOR I=1 TO 15:FOR J=1 TO 15
7130 INPUT # 9,M(I,J)
7140 NEXT J,I
7150 CLOSEIN:GOTO 7010
7200 REM **** CARGAR FORMULAS ****
7210 GOSUB 7500:OPENIN I$
7220 FOR I=1 TO 255
7230 INPUT # 9,F$(I)
7240 NEXT I
7250 CLOSEIN:GOTO 7010
7300 REM **** GUARDAR DATOS ****
7310 GOSUB 7500:OPENOUT I$
7320 FOR I=1 TO 15:FOR J=I TO 15
7330 PRINT # 9,M(I,J)
7340 NEXT J,I
7350 CLOSEOUT:GOTO 7010
7400 REM **** GUARDAR FORMULAS ****
7410 GOSUB 7500:OPENOUT I$
7420 FOR I=1 TO 255
7430 PRINT # 9,F$(I)
7440 NEXT I
7450 CLOSEOUT:GOTO 7010
7500 LOCATE 1,22:INPUT"NOMBRE ARCHIVO";I$
7510 RETURN
```

Sinclair Spectrum:

"H": Imprime pantalla HELP
"E": Entrar datos numéricos
"F": Entrar nueva fórmula
"C": Calcular hoja
"S": Almacenar hoja actual
"G": Tomar hoja almacenada
"Z": Borrar hoja
"R": Reproducir fórmula
"T": Tabular cursor
"D": Rutinas Cargar/guardar

Nota: Pulsar CAPS LOCK antes de la ejecución

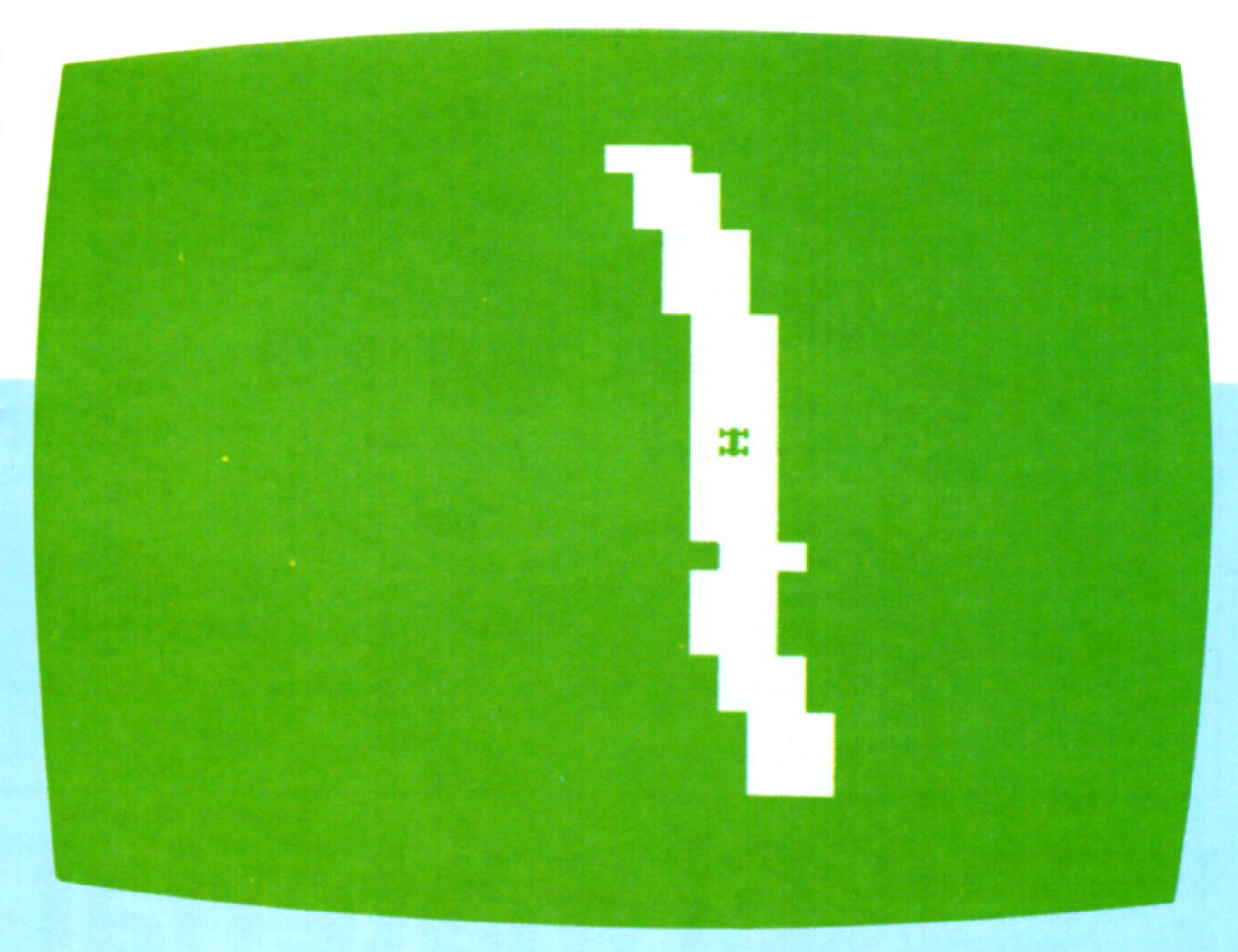
```
6000 > REM IMPRIMIR PANTALLA AYUDA
6010 CLS
6020 PRINT : PRINT " H – IMPRIME ESTA
     PANTALLA HELP"
6030 PRINT : PRINT " E – ENTRAR DATOS
     NUMERICOS"
6040 PRINT : PRINT " F – ENTRAR NUEVA
     FORMULA"
6050 PRINT : PRINT " C – CALCULAR
     HOJA"
6060 PRINT : PRINT " S – ALMACENAR HOJA
     ACTUAL"
6070 PRINT : PRINT " G – TOMAR HOJA
     ALMACENADA"
6080 PRINT : PRINT " Z – BORRAR
     HOJA"
6090 PRINT : PRINT " R – REPRODUCIR
     FORMULAS"
6100 PRINT : PRINT " T – TABULAR A NUEVA
     CELDA"
6110 PRINT : PRINT " D – OPCIONES
     CARGAR/GUARDAR"
6120 LET A$=INKEY$: IF A$="" THEN GO TO
     6120
6130 GO SUB 1000: GO SUB 1700:
     GOTO 1100
7000 REM OPCIONES CARGAR/
     GUARDAR
7010 CLS: PRINT
7020 PRINT " OPCIONES CARGAR Y
     GUARDAR"
7030 PRINT
7040 PRINT : PRINT "            1 – CARGAR
     HOJA DATOS"
7045 PRINT : PRINT "            2 – CARGAR
     MATRIZ FORMULAS"
7050 PRINT : PRINT "            3 – GUARDAR
     HOJA DATOS"
7055 PRINT : PRINT "            4 – GUARDAR
     MATRIZ FORMULAS"
7056 PRINT: PRINT "             X – PARA
     SALIR"
7060 LET A$=INKEY$: IF A$="" THEN GO TO
     7060
7070 IF A$="1" THEN GO TO 7100
7080 IF A$="2" THEN GO TO 7200
7090 IF A$="3" THEN GO TO 7300
7095 IF A$="4" THEN GO TO 7400
7096 IF A$="X" THEN GO SUB 1000:GO SUB
     1700: RETURN
7100 REM **** CARGAR DATOS ****
7110 GO SUB 7500
7120 LOAD I$ DATA M()
7130 GO TO 7010
7200 REM **** CARGAR FORMULAS ****
7210 GO SUB 7500
7220 LOAD I$ DATA F$()
7230 GO TO 7010
7300 REM ***** GUARDAR DATOS *****
7310 GO SUB 7500
7320 SAVE I$ DATA M()
7330 GO TO 7010
7400 GO SUB 7500
7420 SAVE I$ DATA F$()
7430 GO TO 7010
7500 PRINT AT 18,0
7510 INPUT "ENTRAR NOMBRE ARCHIVO";I$
7520 RETURN
```



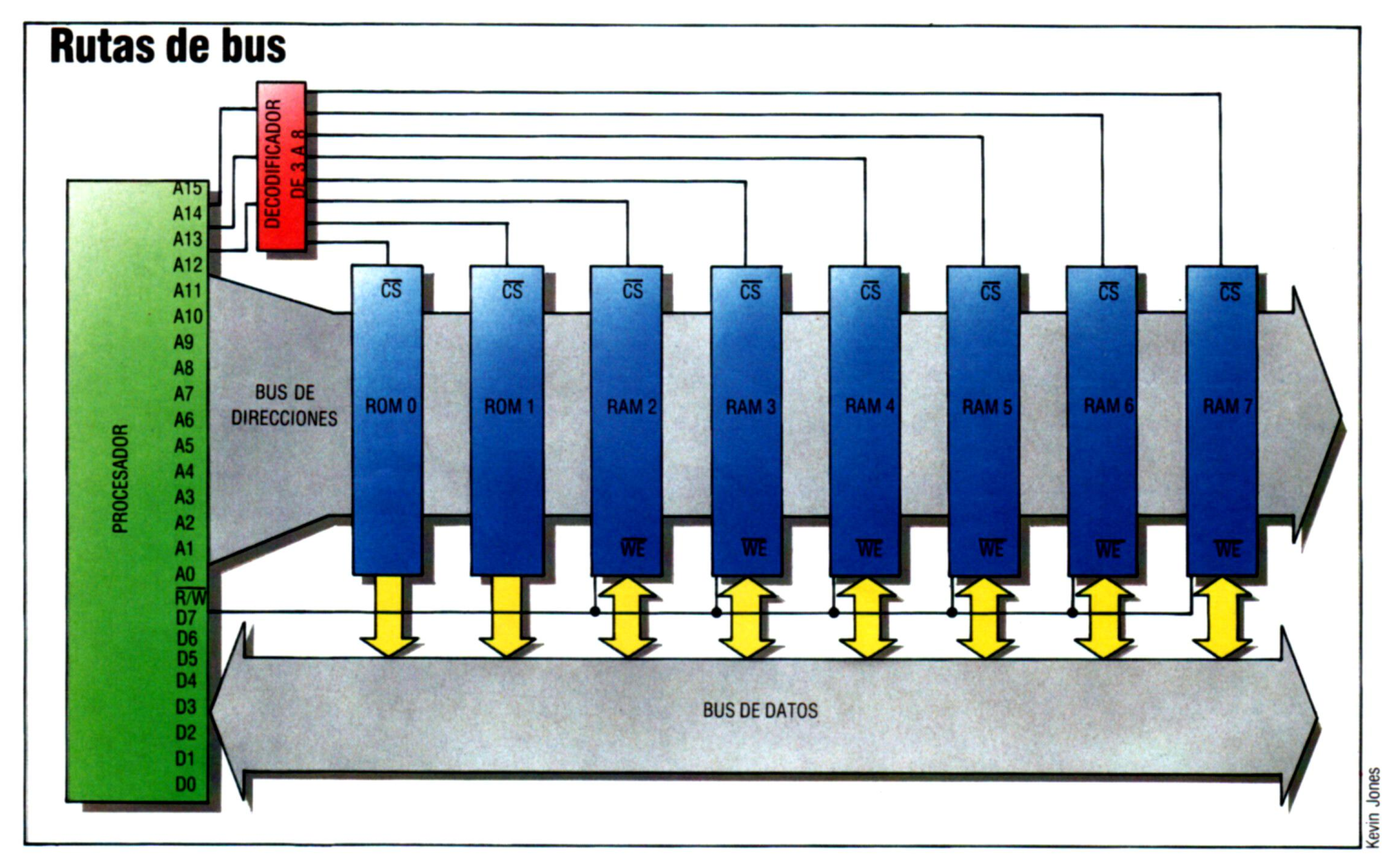
Gran Premio

He aquí nuevamente este popular juego que siempre encabeza los "hit parades" de los programas recreativos. En esta ocasión presentamos una versión para el microordenador MSX

Al volante de su Fórmula 1 intente recorrer la mayor distancia posible. Su coche dispone de dos velocidades. Mantenga pulsada la barra espaciadora para correr en segunda velocidad. La dirección se controla mediante las teclas del cursor. En segunda, el vehículo marcha a doble velocidad. Pero ¡cuidado con los accidentes!



```
10  REM ***************
20  REM * GRAN PREMIO *
30  REM ***************
40  KEY OFF
50  WIDTH 39
60  GOSUB 450
70  H=STICK (0)
80  PX=PX+(H=7)-(H=3)
90  IF STRIG(0) THEN T=2 ELSE T=1
100 IF VPEEK (PX+40)<>CR THEN 240
110 RX=RX+(RND(1)<.5) - (RND (1)<.5)
120 IF RX<RN THEN RX=RN
130 IF RX>RM THEN RX=RM
140 VPOKE XP,CR
150 LOCATE RX,24,0
160 PRINT R$
170 VPOKE PX,V
180 K=K+T
190 DL=(2+T)*50
200 FOR I=1 TO DL
210 NEXT I
220 XP=PX
230 GOTO 70
240 VPOKE XP,CR
250 FOR I=1 TO 5
260 VPOKE PX+40,A
270 FOR J=1 TO 100
280 NEXT J
290 BEEP
300 VPOKE PX+40,V
310 FOR J=1 TO 100
320 NEXT J
330 NEXT I
340 LOCATE 10,10,0
350 PRINT "KMS RECORRIDOS :";K;
360 IF INKEY$<>"" THEN 360
370 LOCATE 14,16,0
380 PRINT "OTRA ?";
390 D$=INKEY$
400 IF D$="" THEN 390
410 IF D$<>"N" AND D$<>"n"
    THEN RUN
420 LOCATE 0,0,1
430 CLS
440 END
450 CLS
460 SCREEN 0,0
470 GOSUB 680
480 DEFINT A-Y
490 COLOR 14,12
500 A=221
510 R$=CHR$ (219) +CHR$ (219) +
    CHR$ (219)
520 CR=219
530 RX=18
540 V=128
550 RY=24
560 T=1
570 PX=420
580 XP=PX
590 LOCATE 0,0,0
600 FOR I=0 TO 24
610 LOCATE RX,I,0
620 PRINT R$
630 NEXT I
640 VPOKE PX,V
650 RM=34
660 RN=2
670 RETURN
680 FOR I=0 TO 7
690 READ A
700 VPOKE 3072+I,A*4
710 NEXT I
720 RETURN
730 DATA 18,0,18,51,51,18,0,18
```



Líneas de escritura
He aquí una disposición típica de un procesador conectado a 64 Kbytes de RAM estática y ROM. La línea R/W (*read/write*) del procesador está conectada con la línea de permisión de escritura, WE *(write entable)*, de cada una de las RAM. Cuando el procesador envía *low* a la línea WE, se pueden escribir datos en la posición de RAM direccionada. Los tres bits superiores simplemente se decodifican en ocho líneas de selección de chip y se utilizan para habilitar una RAM o ROM determinada. Los 13 bits *low* se transportan a la totalidad de los ocho chips de memoria y especifican la posición en cuestión

El material de la memoria

En esta ocasión centraremos nuestra atención en los chips de memoria y veremos cómo se puede escribir o leer un byte

Un punto sobre el hardware del ordenador que la mayoría de las personas tienen claro es la diferencia entre memoria RAM y memoria ROM. La RAM es una memoria que se puede leer y en la que se puede escribir, y la ROM sólo se puede leer, habiéndose definido su contenido durante el proceso de fabricación. Sin embargo, hay dos clases principales de RAM, denominadas *estática* y *dinámica*.

En este capítulo veremos cómo el microprocesador selecciona un byte de memoria determinado para efectuar una operación de lectura o escritura.

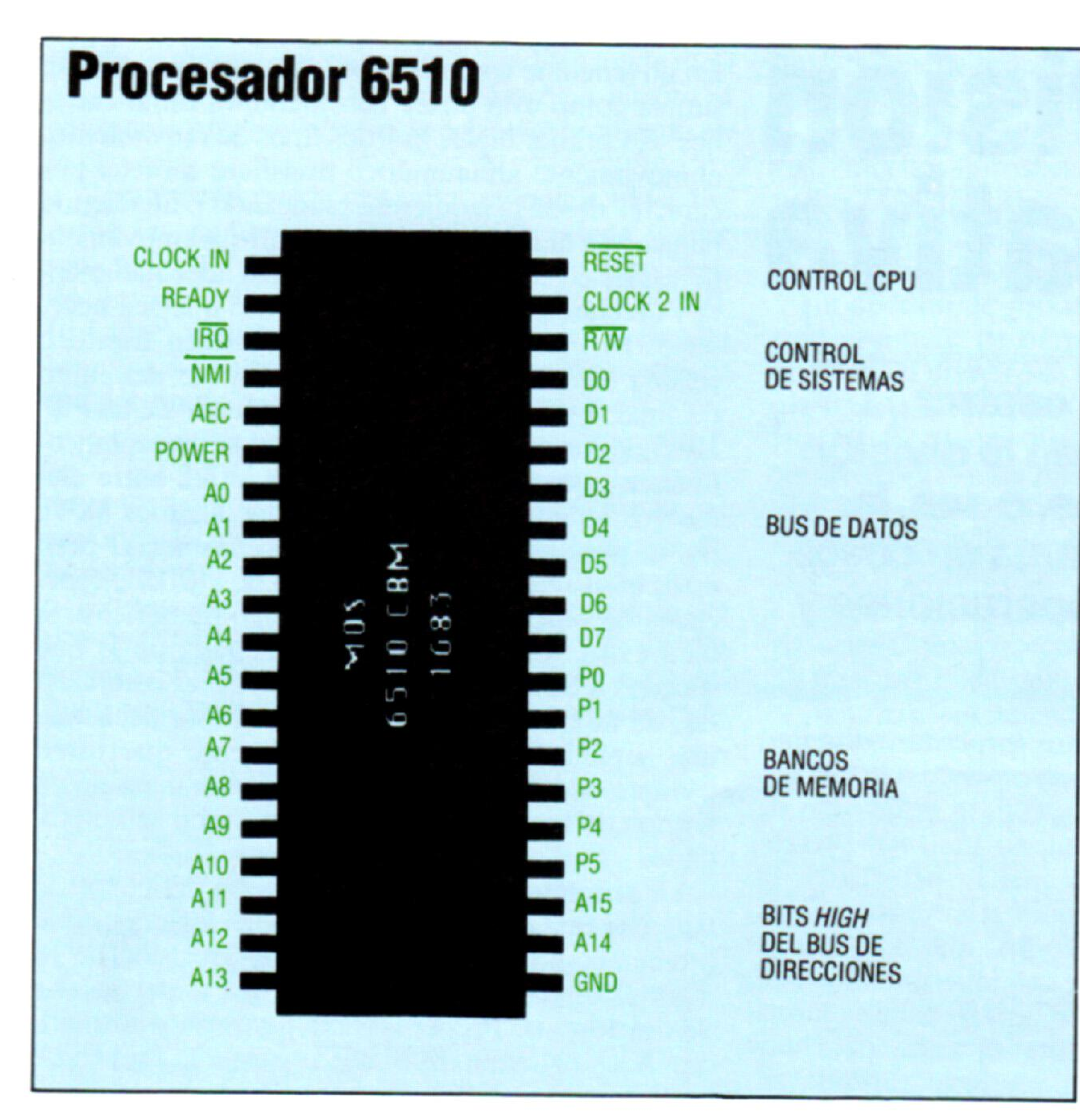
La RAM *estática* utiliza un pequeño circuito lógico secuencial que se conoce como flip-flop J-K para almacenar bits de datos individuales. Un flip-flop posee la capacidad de retener un estado determinado (ya sea cero o uno) en su salida hasta que un impulso lo haga bascular al estado opuesto. Se usan 8 flip-flops para formar cada byte de memoria y, en consecuencia, un chip de RAM estática tendrá empaquetados muchos de tales dispositivos.

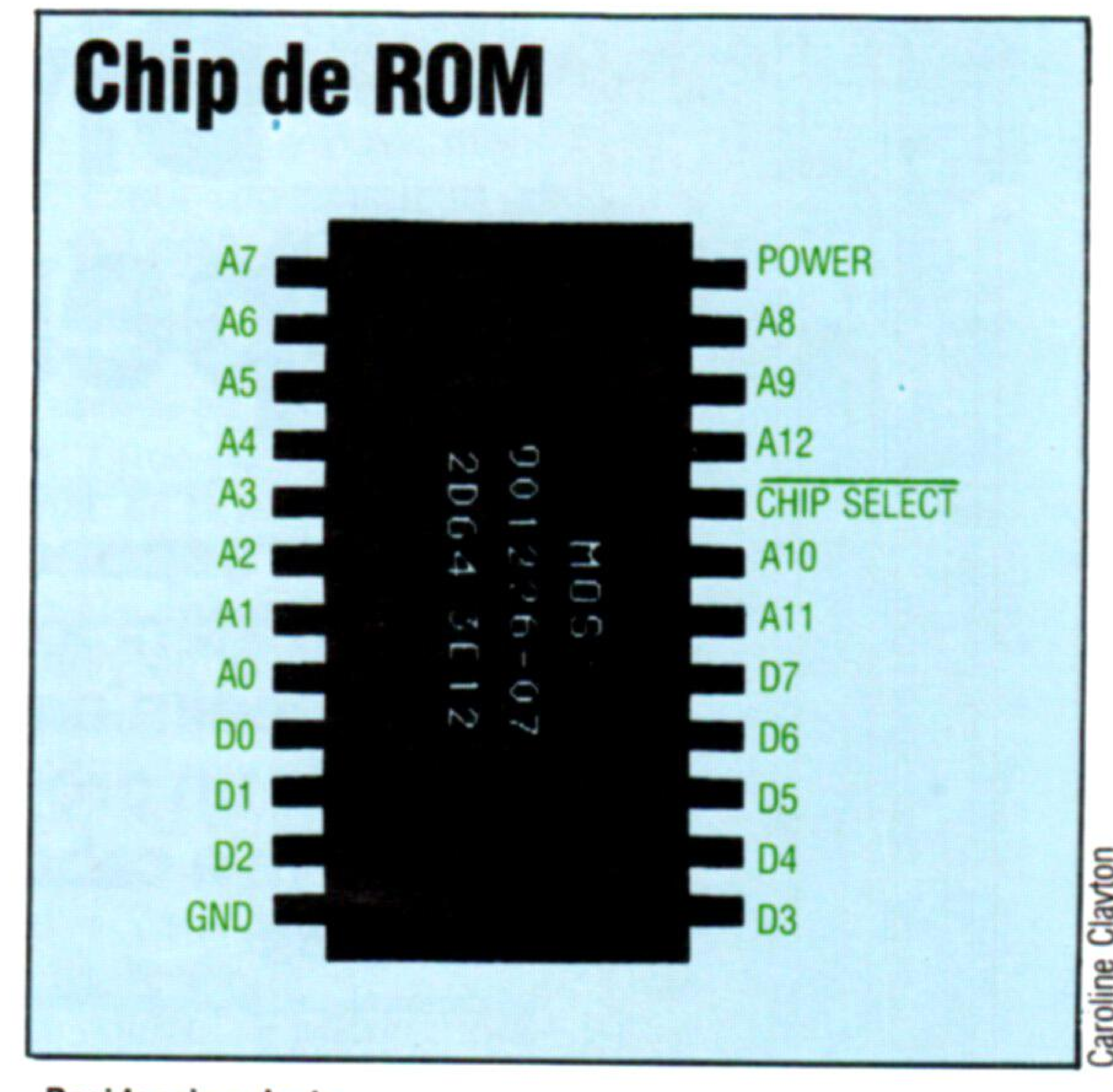
La RAM *dinámica* almacena bits de datos como cargas eléctricas. Aunque almacenar datos de este modo ocupa mucho menos espacio (permitiendo que las RAM dinámicas tengan empaquetada más memoria por chip), las RAM se han de refrescar regularmente porque la carga tiende a debilitarse.

Los diagramas de la página contigua (arriba) muestran las conexiones de patillas para el microprocesador 6510 (un derivado del 6502) utilizado por el Commodore 64 y una típica ROM de 8 K. El procesador posee cuatro grupos de patillas principales: un bus de direcciones de 16 bits (A0 a A15), un bus de datos de ocho bits (D0 a D7), líneas de control de la CPU y del sistema. El 6510 posee la característica inusual de una puerta de seis bits en el chip que permite, entre otras cosas, conmutar bancos adicionales de ROM o RAM en su espacio de direcciones de 64 K.

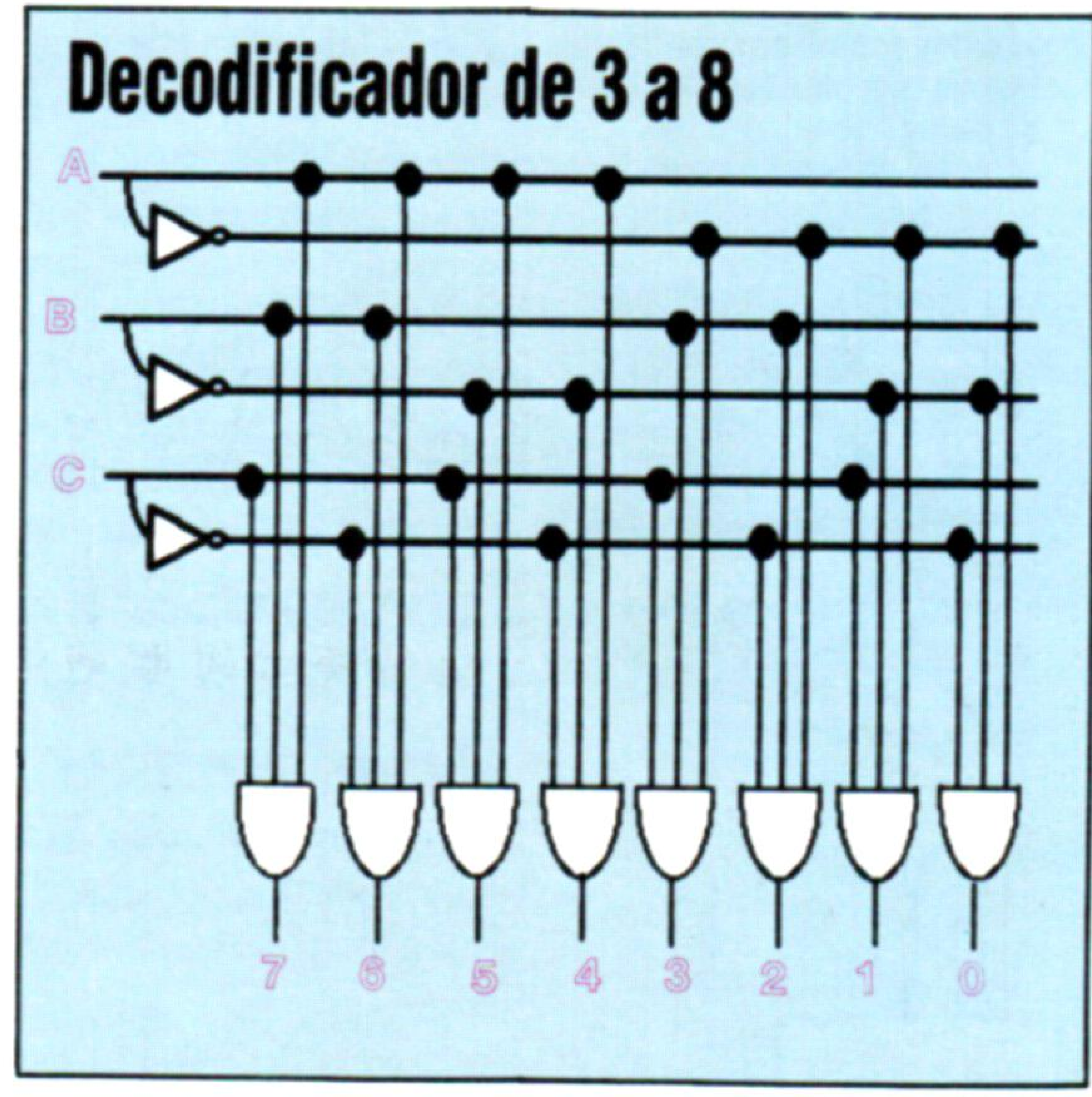
La ROM de 8 K reflejada posee 13 líneas de direcciones (A0 a A12), que le permiten seleccionar 8 192 posiciones individuales, ocho líneas de datos y una patilla de selección de chip. Esta disposición también es típica de un chip de RAM estática, con la excepción de que tal chip de RAM tendría una







Residencia selecta
El chip de ROM de 8 K que vemos aquí posee una disposición de patillas típicas entre esta clase de dispositivos. Se necesitan trece líneas de direcciones para direccionar individualmente cada una de las 8 192 posiciones, ocho patillas de datos permiten la conexión directa al bus de datos y se utiliza una línea de selección de chip junto con un decodificador externo para seleccionar esa ROM en lugar de otras ROM del sistema



Veinte pares de patillas
La mayoría de los procesadores de ocho bits están diseñados de acuerdo a un formato de 40 patillas, como el 6510 que vemos aquí. Observe las 16 patillas de direcciones, las ocho patillas de datos, las señales externas de reloj y control. En el control del sistema se incluyen entradas de interrupciones y una línea de permisión de lectura/escritura. Este procesador es una versión mejorada del 6502, siendo la principal adición un registro de datos en el chip utilizado para E/S de cassette y para controlar los bancos de memoria de los 84 K de RAM y ROM

línea adicional de permisión de escritura. La clave para el direccionamiento de RAM o RAM estática es la línea de selección de chip.

Un procesador de ocho bits posee la capacidad de direccionar hasta 64 K de memoria utilizando su bus de direcciones de 16 bits (puesto que 2↑16=65 536=64 K). Por tanto, el procesador podría acceder a ocho chips de ROM o RAM de 8 K cada uno. Las líneas de dirección de A0 a A12 son necesarias para localizar el byte en el chip de memoria adecuado, pero ello nos deja a A13, A14 y A15, que se pueden utilizar para seleccionar el chip en cuestión. La disposición de, pongamos por caso, dos ROM de 8 K y seis RAM estáticas de 8 K podría ser la que se muestra aquí. Los tres bits superiores de la dirección se pasan por un decodificador de 3 a 8 para seleccionar el chip que contiene el byte direccionado, y los otros 13 bits de la dirección se cablean en común a los 8 chips, para seleccionar el byte concreto del chip seleccionado. Por consiguiente, todas las posiciones cuyas direcciones comiencen con 000 se pueden encontrar en el chip 0; aquellas cuyas direcciones empiecen con 001 se seleccionan en el chip 1, y así sucesivamente.

Mientras que las ROM sólo pueden ser leídas, las RAM se pueden leer o bien grabar. Para seleccionar la operación requerida, todas las RAM poseen una línea de permisión de escritura (WE: *write enable*) conectada a la línea de lectura/escritura (R/W: *read/write*) del procesador. En el caso del 6510, la línea R/W siempre se retiene *high*, excepto cuando se ha de efectuar una operación de escritura, momento en que se pone *low*. Ahora se puede rastrear la acción de los ciclos de lectura o escritura y su incidencia en la memoria.

Cuando el procesador comienza a ejecutar una instrucción READ, como "cargar en el acumulador el contenido de la posición $C000", la dirección ($C000) se coloca en el bus de direcciones y la línea R/W se mantiene *high*. Los tres bits superiores de la dirección serán, en este ejemplo, 110, de modo que el sexto chip de RAM (que se denomina RAM 6) de nuestra disposición de memoria sería seleccionado a través del decodificador de 3 a 8. Los 13 bits restantes son todos cero, de modo que se seleccionará la primera posición de RAM 6 y se conectará al bus de datos de modo que su contenido se pueda volver a transmitir al procesador y colocar en el acumulador.

Durante una instrucción WRITE (como "almacenar en $C000 el contenido del acumulador") la posición se selecciona de la misma manera pero esta vez la R/W se mantiene *low*, haciendo que la patilla de permisión de lectura (WE) se ponga *low*. En esta etapa de la instrucción el contenido del acumulador ya se habrá colocado en el bus de datos y estará escrito en la primera posición de RAM 6.

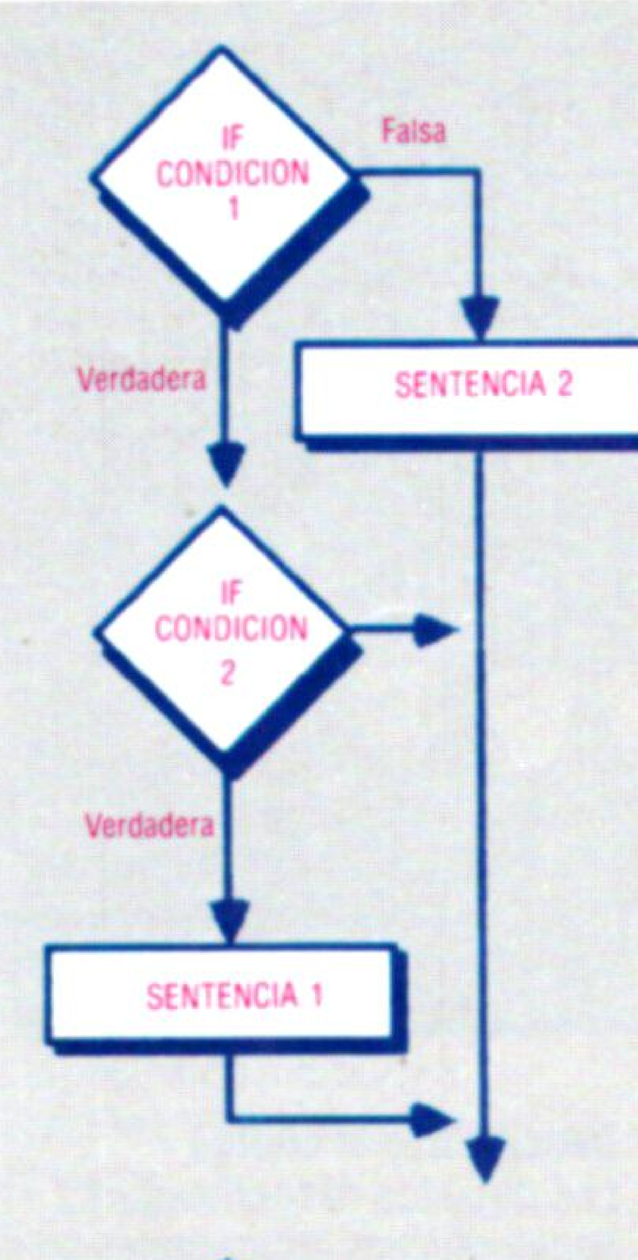
Descifrando el código
Los circuitos decodificadores tienen muchas aplicaciones en el hardware de ordenador. Su labor consiste en aceptar una entrada codificada en binario y producir señales individuales que correspondan a cada permutación del código. Por consiguiente, en ocho salidas separadas se puedcn decodificar tres entradas; la línea 0 corresponde a 000 en las entradas, la línea 1 corresponde a 001 y así sucesivamente hasta la línea 7, que es activada por 111 en las líneas de entrada

División operativa

Nos corresponde centrar nuestra atención en la división de procedimientos, o sea, la parte de un programa en COBOL que lleva a cabo operaciones y cálculos

La división de procedimientos (*procedure division*) de un programa en COBOL corresponde a la función básica del programa. El trazado de la división de procedimientos y el vocabulario utilizado pretenden conseguir que el programa se parezca en la mayor medida posible al inglés empresarial. Cada sección se compone de párrafos, que a su vez se componen de oraciones que deben terminar con un punto y aparte. Una oración correspondería a una sentencia en un lenguaje como el BASIC, pero una oración COBOL puede a su vez estar dividida en cláusulas (las comas son opcionales), cada una de las cuales puede ser una "sentencia" o calificar a otra. Se dice que la palabra clave de una cláusula o una oración que especifica la acción a emprender es el *verbo*. Con mucho, el verbo más utilizado es MOVE, que se limita a transferir datos de una zona a otra, tomando la forma:

```
MOVE identificador-1 TO identificador-2.
```



Problemas con los IF

El empleo del punto y aparte como terminador para la construcción IF en COBOL puede plantear problemas. Por ejemplo, la construcción:

```
IF condición1 (then)
   IF condición 2 (then)
      sentencia1
ELSE condición2.
```

se ejecutaría como podemos ver en el diagrama de flujo superior. No obstante, el papel del punto y aparte como terminador de AMBAS sentencias significa que el COBOL compila la alternativa no deseada que vemos abajo. La forma de sortear este problema consiste en colocar la selección interior en un párrafo separado y después llevar a cabo (PERFORM) ese párrafo. De modo que la construcción de arriba se codificaría mejor en COBOL como:

```
IF condición1 (then)
   PERFORM párrafo 1
   ELSE sentencia2
......
párrafo1
   IF condición2
      sentencia1
```

En un lenguaje como el COBOL, incluso una idea tan simple como ésta posee considerables ramificaciones. En primer lugar, hay dos tipos de movimiento: el movimiento alfanumérico transfiere carácter por carácter desde la izquierda, truncando o añadiendo blancos en la medida de lo necesario; el movimiento numérico coloca el punto decimal adecuadamente y efectúa cualquier cambio de uso que sea necesario. El movimiento numérico también trunca o rellena con ceros ya sea antes o después del punto decimal. El tipo de movimiento depende de las PICTURE de los dos datos. En realidad la diversión comienza cuando uno considera los MOVE entre elementos alfanuméricos y numéricos; algunos MOVE no se permiten bajo ninguna circunstancia, pero éstos básicamente implican el uso de PICTURE especiales de edición y MOVE alfabético-numéricos. Si bien existe una especificación detallada de lo que sucede en cada MOVE legítimo, aun así es complicado, de modo que por el momento basta decir que una sentencia que permita fácilmente que usted trunque dígitos significativos sin ningún mensaje en tiempo de ejecución se debe utilizar con sumo cuidado.

La aritmética no es el punto fuerte del COBOL, si bien hay que decir que son pocos los lenguajes que ofrecen como facilidad estándar la precisión de 18 dígitos del COBOL. La aritmética se puede realizar de dos formas. Primero, están los verbos aritméticos ADD (sumar), SUBTRACT (restar), MULTIPLY (multiplicar) y DIVIDE (dividir). P. ej.:

```
ADD número-1 TO número-2.
```

suma el contenido de número-1 al contenido de número-2, y:

```
ADD número-1 TO número-2 GIVING número-3.
```

forma la suma de número-1 y número-2 en número-3. Existe otra opción que permite llevar a cabo la misma aritmética en varios lugares:

```
ADD 5 TO número-1,número-2
```

p. ej., le suma 5 a los valores actuales tanto de número-1 como de número-2.

Los otros verbos aritméticos responden al mismo formato general:

```
SUBTRACT número-1 FROM número-2 [GIVING
   número-3].
MULTIPLY número-1 BY número-2 [GIVING
   número-3].
DIVIDE número-1 INTO número-2
   [GIVING número-3]
   [REMAINDER número-4].
```

Observe, en cada caso, que el valor final se halla en el campo enumerado en último lugar.

Al igual que con los MOVE, se plantean problemas cuando el resultado de una operación aritmética es demasiado largo para el espacio que se ha preparado para almacenarlo. Si se pierden dígitos menos significativos, normalmente se truncarán, excepto cuando el nombre del dato receptor va seguido de la palabra ROUNDED (redondeado). Truncar los dígitos más significativos es, por supuesto, mucho más peligroso. El COBOL permitirá que esto suceda, pero da la opción de añadir una cláusula ON SIZE ERROR (en caso de error de tamaño) a cualquier sentencia aritmética. Esto no tiene ningún



efecto, a menos que se produzca un recorte de dígitos más significativos, en cuyo caso deja intacto el contenido previo y efectúa la acción especificada en la cláusula.

Como usted puede imaginar, en COBOL todo, excepto la aritmética más simple, se puede volver muy tedioso de escribir. Las cosas se han simplificado gracias a la introducción del verbo COMPUTE (calcular), que puede ir seguido por una sentencia aritmética de una forma similar a una sentencia de asignación de BASIC o FORTRAN, como:

COMPUTE número-1=(número-2 + 3)* número-3.

Algunos compiladores incluso permiten el uso de exponenciación en tales expresiones, pero ello no es estándar.

Existen varios problemas que surgen a raíz del uso de COMPUTE, uno de los cuales es el problema de SIZE ERRORS que se podrían producir en cualquier resultado intermedio así como en el resultado final. Se puede utilizar una cláusula ON SIZE ERROR, pero la misma no le dirá en qué etapa del cálculo se produjo el desbordamiento. Por este y otros motivos, muchos puristas del COBOL no emplean el verbo COMPUTE sino que se basan en los verbos aritméticos comunes, que pueden ser tediosos de escribir pero resultan mucho más claros y seguros.

La división de procedimientos se prepara en un cierto número de párrafos. Cada párrafo lleva un nombre; el nombre se escribe en la zona A que empieza en la columna 7 y las sentencias se escriben en la zona B, que empieza en la columna 12. Estos párrafos operan de forma muy similar a una subrutina de BASIC. En consecuencia, por lo general no hay módulos ni procedimientos separados como en PASCAL, y no hay ni parámetros ni variables locales. La ejecución normal del código es desde el comienzo hasta el final, ejecutándose cada párrafo de uno en uno a medida que van apareciendo.

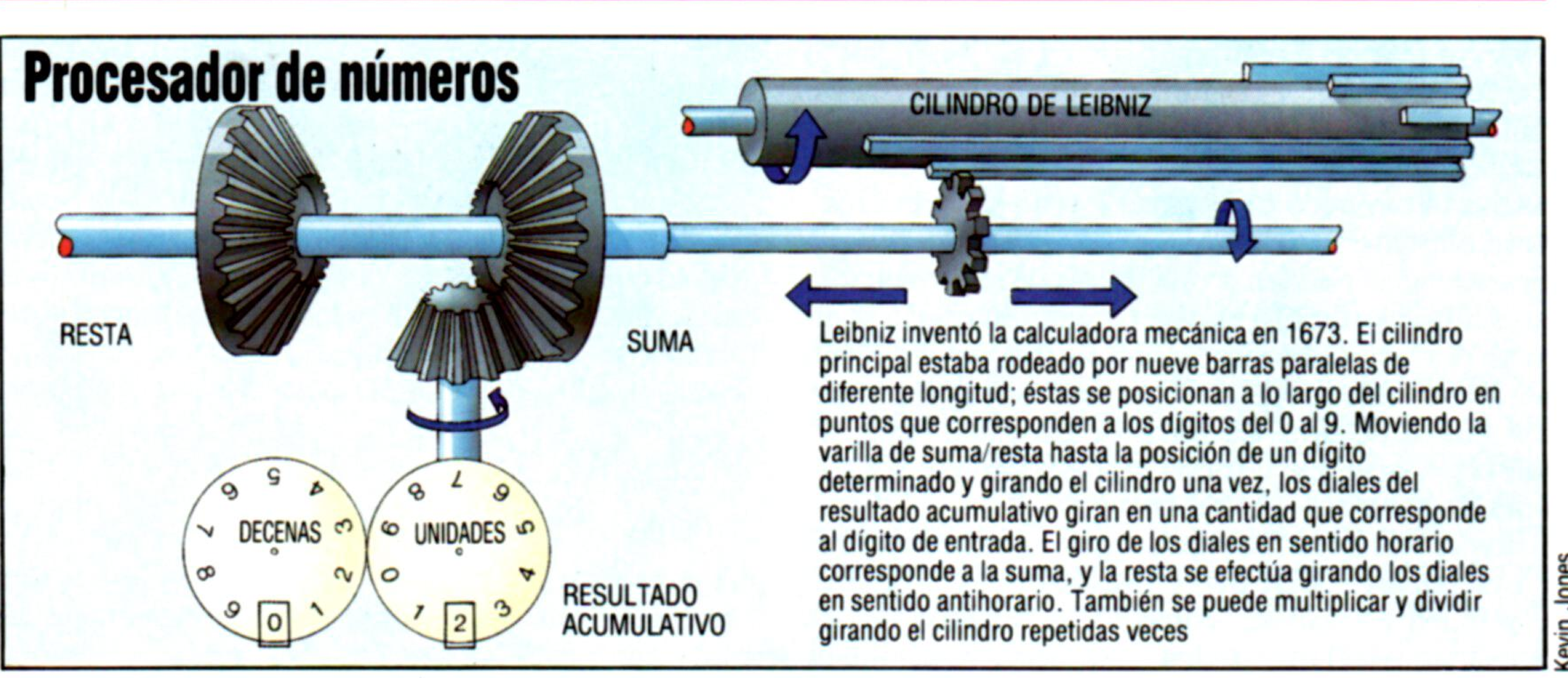
En COBOL la secuencia de control se puede modi-

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PI-CALC.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES. CONSOLE IS CRT.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
*
 01  SCREEN PIC X(1920).
*
 01  DI-1 REDEFINES SCREEN.
     02 FILLER PIC X(160).
     02 DI-TX1 PIC X(160).
     02 DI-TX2 PIC X(13).
     02 DI-TERM PIC X(15).
     02 FILLER PIC X(136).
     02 DI-TX3 PIC X(6).
     02 DI-PI PIC X(15).
     02 FILLER PIC X(1415).
*
 01  DI-2 REDEFINES SCREEN.
     02 FILLER PIC X(333).
     02 DI-TERM2 PIC X(15).
     02 FILLER PIC X(142).
     02 DI-PI2 PIC X(15).
     02 FILLER PIC X(1415).
*
 01  WORK-AREA.
     02 PI PIC S9V9(14).
     02 TERM PIC S9V9(14).
     02 W PIC S9V9(14).
     02 N PIC 9999.
     02 N1 PIC 9999.
     02 N2 PIC 9999.
     02 ED PIC -9.9(12).
*
 01  CONSTANTS.
     02 TX1 PIC X(17) VALUE "CALCULATION OF PI".
     02 TX2 PIC X(12) VALUE "NEXT TERM IS".
     02 TX3 PIC X(5) VALUE "PI IS".
*
PROCEDURE DIVISION.
LA-START.
    DISPLAY SPACE.
    MOVE SPACE TO SCREEN.
    MOVE TX1 TO DI-TX1.
    MOVE TX2 TO DI-TX2.
    MOVE TX3 TO DI-TX3.
    MOVE 0.5 TO ED.
    MOVE ED TO DI-TERM.
    MOVE 3 TO ED.
    MOVE ED TO DI-PI.
    DISPLAY DI-1.
    MOVE 0.5 TO PI.
    MOVE 0.5 TO TERM.
    MOVE 3 TO N.

LOOP.
    MOVE N TO N2.
    SUBTRACT 2 FROM N2.
    MULTIPLY N2 BY N2.
    MULTIPLY N2 BY TERM.
    MOVE N TO N1.
    SUBTRACT 1 FROM N1.
    MULTIPLY N BY N1.
    MULTIPLY 4 BY N1.
    DIVIDE N1 INTO TERM.
    IF TERM < 0.000000000001 THEN GO TO HALT.
    ADD TERM TO PI.
    MOVE PI TO W.
    MULTIPLY 6 BY W.
    MOVE W TO ED.
    MOVE ED TO DI-PI2.
    MOVE TERM TO ED.
    MOVE ED TO DI-TERM2.
    DISPLAY DI-2.
    ADD 2 TO N.
    IF N < 100 GO TO LOOP.
HALT.
    STOP RUN.
```

Calculando "pi"
Este programa en COBOL ilustra el uso de los verbos aritméticos para calcular un valor para *pi* usando las series de Leibniz. Esta serie de expresiones algebraicas opera sobre el principio de calcular una serie infinita de términos, con cada término de la serie volviéndose sucesivamente más pequeño.
Gran parte del trabajo de Leibniz ha sido de suma importancia para los procedimientos informáticos modernos y, aun en su propio tiempo, condujo al desarrollo de "máquinas de calcular", una de las cuales se ilustra abajo. El programa en COBOL, ejecutado en una máquina bastante más moderna, se escribió bajo CP/M utilizando MicroFocus CIS-COBOL

Procesador de números



Leibniz inventó la calculadora mecánica en 1673. El cilindro principal estaba rodeado por nueve barras paralelas de diferente longitud; éstas se posicionan a lo largo del cilindro en puntos que corresponden a los dígitos del 0 al 9. Moviendo la varilla de suma/resta hasta la posición de un dígito determinado y girando el cilindro una vez, los diales del resultado acumulativo giran en una cantidad que corresponde al dígito de entrada. El giro de los diales en sentido horario corresponde a la suma, y la resta se efectúa girando los diales en sentido antihorario. También se puede multiplicar y dividir girando el cilindro repetidas veces

ficar de numerosas maneras, básicamente mediante el verbo PERFORM (efectuar), que posee muchas variantes. Dos de éstas son:

```
PERFORM párrafo-1.
```

que actúa como una llamada a procedimiento o como un GOSUB, ejecutando las sentencias del párrafo y luego devolviendo el control a la sentencia que sigue al PERFORM, y

```
PERFORM párrafo-1 THRU párrafo-n.
```

que ejecutará varios párrafos consecutivos antes de devolver el control.

Otra forma de incidir en el flujo de control es utilizar:

```
GOTO párrafo-1.
```

que funciona del modo habitual, y que en COBOL se debe evitar en la misma medida que en otros lenguajes.

Como lenguaje destinado prioritariamente a aplicaciones de gestión, el COBOL proporciona un amplio conjunto de opciones de lectura y escritura para archivos en disco o cinta. Sin embargo, el COBOL estándar tiene facilidades mínimas para entrada/salida interactivas utilizando una pantalla y un teclado. Los dos verbos para manipular esto son:

```
ACCEPT nombre-dato.
```

para entrada desde el teclado, y:

```
DISPLAY nombre-dato.
```

Éstas operan en la modalidad de teletipo normal, de modo muy similar al INPUT y PRINT del BASIC. La mayoría de las versiones de COBOL para micros incluyen verbos ACCEPT (aceptar) y DISPLAY (visualizar) mejorados con el fin de permitir el trabajo en pantalla.

Una de las principales facilidades que se espera de cualquier lenguaje de programación moderno es la gama de estructuras de control disponibles, en especial las facilidades para selección e iteración. El COBOL proporciona una construcción IF...THEN...ELSE para la selección, si bien es limitado en comparación a la que ofrece, por ejemplo, el PASCAL. El formato es:

```
IF condición
   sentencia(s)
ELSE
   sentencia(s).
```

Observe que no hay ningún THEN y también que la sentencia IF acaba con un punto y aparte como cualquier otra sentencia, lo que significa que no puede haber ningún punto y aparte entre las sentencias ni tampoco en la parte IF o en la parte ELSE. Normalmente esto no tiene importancia, porque las sentencias pueden ir una detrás de la otra en una misma oración, pero establece una diferencia si alguna de las sentencias ofrece alternativas, como puede ser otra sentencia IF o READ que compruebe una marca de final de archivo. En estos casos, el punto y aparte que termina la selección interior terminará asimismo la exterior.

Las condiciones booleanas que comprueba el IF se pueden formar de numerosas maneras. Primero, se pueden establecer (como en la mayoría de los otros lenguajes) usando los operadores relacionales <, > y =, así como usando NOT para crear los otros, como en NOT <, que significa lo mismo que > =. Los operadores se pueden escribir como símbolos o bien escribir al completo, como en número-1 IS EQUAL TO 5. Cualquier número de estas condiciones se puede unir con AND y OR para producir una condición compuesta, como en (número-1 = 5) OR (número-2 NOT = 6).

El COBOL es uno de los pocos lenguajes que permiten abreviar estas condiciones compuestas cuando en una condición compuesta se está utilizando nuevamente un nombre de dato o el mismo operador. Las siguientes son dos condiciones de COBOL legales:

```
número-1 < 10 AND > 6
letra-1="A" OR "B" OR "C"
```

El COBOL no posee un tipo de datos booleano, pero proporciona una entrada de datos especial de nivel 88 que se puede asociar con cualquier dato elemental para especificar un valor, escala o conjunto de valores y un nombre de condición booleana que se puede utilizar allí donde se emplearía una condición normal.

Por ejemplo:

```
77 número-1 PIC 99.
88 el-número-es-válido VALORES 1,3,24 THRU 67.
```

(Observe que algunos COBOL sólo permiten ya sea una escala o bien una lista de valores, pero no ambas.)

El nombre de condición el-número-es-válido se asocia con número-1 sólo por estar escrito inmediatamente después de él en la división de datos. Siempre que se almacene un valor en número-1, el valor de el-número-es-válido se ajusta como verdadero o falso adecuadamente y se puede comprobar de forma directa, como en:

```
IF el-número-es-válido
   PERFORM rutina-válido
ELSE
   PERFORM rutina-no-válido.
```

En COBOL, la iteración (efectuar un bucle) se puede realizar de diversas maneras, todas las cuales implican variantes de la sentencia PERFORM. La forma básica es:

```
PERFORM nombre-párrafo UNTIL condición.
```

donde la condición puede ser cualquier combinación de relaciones o elementos de nivel 88, como con IF. Actúa como un bucle WHILE de BASIC y otros lenguajes, comprobándose la condición cada vez antes de llevar a cabo el párrafo. Si la condición comienza siendo verdadera, el párrafo no se efectúa nunca.

En COBOL, el equivalente más próximo a un bucle FOR...NEXT de BASIC es una variante de este uso del PERFORM, que permite incrementar uno o más datos numéricos a partir de un valor inicial y en función de un valor de paso. La siguiente sentencia llevará a cabo el código de párrafo-1 cinco veces:

```
PERFORM párrafo-1 VARYING número-1 FROM
   1 BY 1
UNTIL número-1>5.
```

Observe que la condición terminadora puede ser cualquiera: no ha de implicar necesariamente al dato numérico.

TRABAJAR CON PALABRAS

En esta nueva serie repasaremos los conceptos fundamentales del tratamiento de textos y analizaremos algunos de los paquetes más populares



Chris Stevens

Verdaderamente especializado
El Amstrad PCW 8256 es el último de la gama de productos de precio reducido de la firma Amstrad. Empaquetado como procesador de textos especializado y basado en el chip Z80 con 256 Kbytes de RAM disponible, el PCW proporciona un ordenador, pantalla, unidad de disco integrada, impresora y software para tratamiento de textos empaquetado

El tratamiento de textos es, esencialmente, una combinación de distintas operaciones, algunas de las cuales dependen del usuario y otras las realiza completamente el ordenador. Estas operaciones son la entrada de texto en el ordenador, la creación de un archivo en RAM o en disco para almacenar el texto (por lo general, en formato ASCII), la visualización de todo el archivo o parte del mismo en una VDU de forma legible, la manipulación de los datos de ese archivo y, por último, el tratamiento de los diversos archivos creados.

Es necesario comprender todo esto de forma clara, porque todo procesador de textos se ha de juzgar en función del éxito con el que lleva a cabo, o permite que el usuario lleve a cabo, estas operaciones. Algunos programas de tratamiento de textos son, por ejemplo, especialmente buenos para visualizar texto (*MacWrite*, p. ej., muestra en pantalla toda una variedad de exóticas fuentes), mientras que otros son soberbios para el tratamiento de archivos. Además del rendimiento de estas tareas de nivel relativamente bajo, el software para tratamiento de textos también se debe juzgar en función de la potencia y las facilidades de alto nivel que ofrezca.



Cortesía de Microwriter Ltd.

Lenguaje de dedos
El Microwriter ofrece un concepto completamente diferente en tratamiento de textos al de cualquier otra máquina disponible en el mercado. Prescindiendo por completo del teclado QWERTY, los caracteres individuales se entran pulsando distintas combinaciones de teclas. Una vez entrado el texto de un documento, el Microwriter se conecta, entonces, a una impresora para obtener una copia impresa

En la actualidad el tratamiento de textos se ha vuelto tan familiar que la compleja programación que requiere se suele dar por sentada. Además, es un área de programación cuyo origen es bastante reciente. Ello se debe a diversas razones. En primer lugar, es esencial una VDU para visualizar el texto, y éstas sólo se han vuelto dispositivos de salida estándares para ordenadores en los diez últimos años. En segundo lugar, los ordenadores están diseñados para tratar números, no texto. La introducción del código ASCII evidentemente soluciona este problema, pero el código retiene 255 caracteres diferentes y, como tal, no es en absoluto eficiente en cuanto a su uso de memoria. Un teclado de máquina de escribir estándar contiene poco más de 100 caracteres (contando las teclas con 2 caracteres), de modo que es fácil ver que un código de 255 caracteres posee una considerable redundancia incorporada. Por este motivo, el tratamiento de textos no sólo es complejo, sino que también es muy exigente desde el punto de vista de la memoria.

Esta complejidad conduce al problema de tener que abordar el tratamiento de textos de muchas maneras diferentes. Los procesadores de textos vienen en muchas formas, tamaños y formatos diferentes.

Los procesadores de textos especializados, como los que fabrican Wang e IBM, son muy populares en el mercado de gestión. Consisten en un terminal de visualización, un teclado con varias teclas diseñadas específicamente para funciones de tratamiento de textos (una tecla Paste [pegar], p. ej.) y un medio para almacenar archivos de texto en disco rígido o flexible. Con la excepción del Amstrad PCW 8256, tales máquinas son caras. No obstante, ofrecen ventajas derivadas de la especialización: teclados construidos especialmente, facilidades para redes y visualizaciones de diseño ergonómico. El tratamiento de textos supone mucho tiempo contemplando la pantalla, de modo que es esencial una visualización ancha (más de 80 columnas) y nítida.



En compañía del ordenador
Los ordenadores portátiles para regazo están ganando terreno gradualmente en el mercado imformático. La ventaja de estas máquinas es que el usuario puede llevar a cabo una "informática sobre la marcha". La mayoría de los ordenadores para regazo, como el Olivetti M10 que vemos fotografiado aquí, tienen retenida en ROM la capacidad para tratamiento de textos. Si bien en estas máquinas las facilidades de tratamiento de textos y la cantidad de memoria libre son restringidas debido a la limitada potencia disponible, son herramientas útiles para cartas, memorándums, etc.

Por el contrario, el software para tratamiento de textos se puede clasificar en varias categorías diferentes. En el nivel inferior, se puede conseguir cierta forma de tratamiento de textos utilizando un sencillo editor de pantalla como el que proporcionan la mayoría de los sistemas operativos. Usted puede emplear esta facilidad para imprimir párrafos cortos en modalidad directa, pero obviamente esto no será suficiente para nada, con la excepción del memorándum más breve.

Ascendiendo en la clasificación está el "programa para tratamiento de textos", que permite que el usuario prepare un archivo de texto y lo edite directamente en pantalla. Éste es un concepto que se acerca mucho más al de un auténtico procesador de textos, y algunos manipuladores de textos incluso ofrecen rutinas de búsqueda (para localizar una serie de caracteres determinada) y funciones de recortar/pegar, que permiten que el usuario manipule pequeños bloques de textos dentro de un archivo. Un buen ejemplo de tales programas es la facilidad TEXT que se ofrece en la gama de ordenadores portátiles Kyocera, comercializados bajo el logotipo del Olivetti M10 y el Tandy Modelo 100. El programa ofrece funciones de recortar, pegar, copiar, buscar e imprimir, almacenando el texto como un archivo en RAM CMOS, con una batería de apoyo, de modo que seguirá estando allí la próxima vez que se use la máquina.

Por último tenemos los programas que pueden reclamar con toda justicia el título de procesadores de textos. Éstos se pueden clasificar en dos categorías: basados en RAM y basados en disco. La primera categoría se introdujo en realidad para beneficio de los usuarios de ordenadores personales que carecían de almacenamiento en disco y, por lo tanto, requerían un programa que se cargara enteramente en RAM y, debido a que el almacenamiento en cassette es tan lento, almacenara también en RAM todo el documento. Sólo cuando éste se ha editado en una versión final es guardado en cinta. Los programas basados en RAM son limitados en cuanto a la gama de facilidades que ofrecen, por razones obvias. En particular, el tamaño de los documentos está severamente limitado, y también suele estar limitado el tamaño de los bloques de texto que se puedan desplazar.

Sin embargo, los programas basados en RAM ofrecen una gran ventaja respecto a sus parientes basados en disco: tienden a ser de operación mucho más rápida, dado que las distintas facilidades no dependen de numerosos accesos a disco para funcionar. Un buen ejemplo de procesador de textos basado en RAM es el *Tasword*, que permite que el usuario cree un documento de hasta 13 000 caracteres, o de algo más de 2 000 palabras.

Los programas basados en disco suelen permitir el trabajo con documentos muchísimo más grandes y, en teoría, las dimensiones del documento sólo están limitadas por la zona de almacenamiento disponible en el disco de datos. El programa carga porciones del archivo en cuestión en la medida en que así se requiera, si bien a menudo esto puede conducir a considerables demoras, especialmente si usted se halla al final de un documento largo y desea retornar al principio. Ejemplos típicos de sistemas basados en disco son el *WordStar* y el *PerfectWriter*. Tales programas con frecuencia vienen empaquetados en sistemas de gestión, pero de lo contrario tienden a ser muy caros. Una ventaja de los programas basados en disco es que suelen hacer un uso exhaustivo del sistema operativo de disco sobre el que operan (CP/M, MS-DOS, etc.) y, en consecuencia, tienden a gozar de una mayor compatibilidad de archivos que sus equivalentes para ordenadores personales.

En esta serie veremos algunos tipos diferentes de procesadores de textos y evaluaremos sus puntos fuertes y sus puntos débiles. Empezando por el *WordStar*, un programa que ha conseguido establecerse como un estándar de la industria en función del cual se juzgan todos los otros programas, continuaremos con el *MacWrite* como ejemplo de un programa que sobresale por su visualización. Asimismo, analizaremos diversos programas basados en RAM, como el *Tasword* y el *Mini-Office*.



Una obra maestra

Su gran capacidad de direccionamiento hace que el microprocesador 68000 pueda competir en potencia hasta con los ordenadores centrales

Un chip para el futuro
La serie 68000 otorga a los micros de hoy la potencia de proceso de un mini, y los fabricantes se han apresurado a sacar partido de ello. Las máquinas que aquí se muestran emplean todas el 68000 y se han desarrollado potentes chips periféricos para tratar gráficos y sonido que se ejecutan en conexión con el chip

Atari 520ST

Sinclair QL

Apple Macintosh

Apple Lisa

El microprocesador 68000 tiene su origen en la serie 6800 de microordenadores de Motorola, tan bien acogida en el mercado, cuyo punto final fue el microprocesador 6809. Motorola reanudó este cabo con su serie 68000 de alto rendimiento, que ha sido posible gracias a los adelantos en técnicas de VLSI (*very large scale integration*: integración a muy grande escala), que permiten albergar más de 100 000 elementos electrónicos lógicos (tales como puertas AND/OR, flip-flops, etc.) en un solo chip de 64 patillas y de apenas 5 cm de longitud. Con esto se consigue una capacidad de direccionamiento muy similar a la del miniordenador DEC PDP-11, con todas las ventajas y la potencia del juego de instrucciones de un ordenador central: todo en un solo chip.

Estudiaremos más adelante esta capacidad de direccionamiento y de instrucciones. De momento, nos vamos a ceñir a la arquitectura global del 68000 para disponer de un modelo útil donde aplicar nuestros programas. La idea de un modelo es importante porque necesitamos limitar el campo de nuestros conocimientos del microordenador a aquellos aspectos que harán que nuestros programas funcionen. Por ejemplo, no necesitamos comprender cómo trabajan los elementos flip-flop, para poder sumar dos números y establecer el resultado en un registro. Pero sí que nos es necesario saber detalles sobre la longitud de un registro y las instrucciones ADD si queremos resolver ese problema. Nuestro modelo queda, por ello, limitado a las funciones imprescindibles para programar un micro.

La arquitectura del hardware

Si examinamos las funciones eléctricas de las patillas del microprocesador 68000 veremos que las señales están relacionadas con el control de los buses digitales. Estos buses son los medios de que se vale el microordenador para comunicarse con el mundo electrónico de su entorno y con los cuales se puede construir un sistema de microordenadores. Sobre una única placa es posible construir un típico sistema como el que se esquematiza en el dibujo. Aquí el procesador puede tomar la instrucción siguiente desde la memoria a través del bus, ejecutar la instrucción internamente dentro del procesador y, si fuera necesario, enviar un carácter a un chip de interface para visualizarlo ante el usuario.

Esta arquitectura del hardware constituye la base de nuestro modelo. Todavía necesitamos pensar

El modelo básico
El esquema muestra, simplificado, un sistema de ordenador típico de una sola placa. Las instrucciones y los datos se toman de la memoria a través del bus y se procesan. Los resultados se depositan de nuevo en el bus para ser transmitidos a la pantalla, a otros periféricos o a la RAM

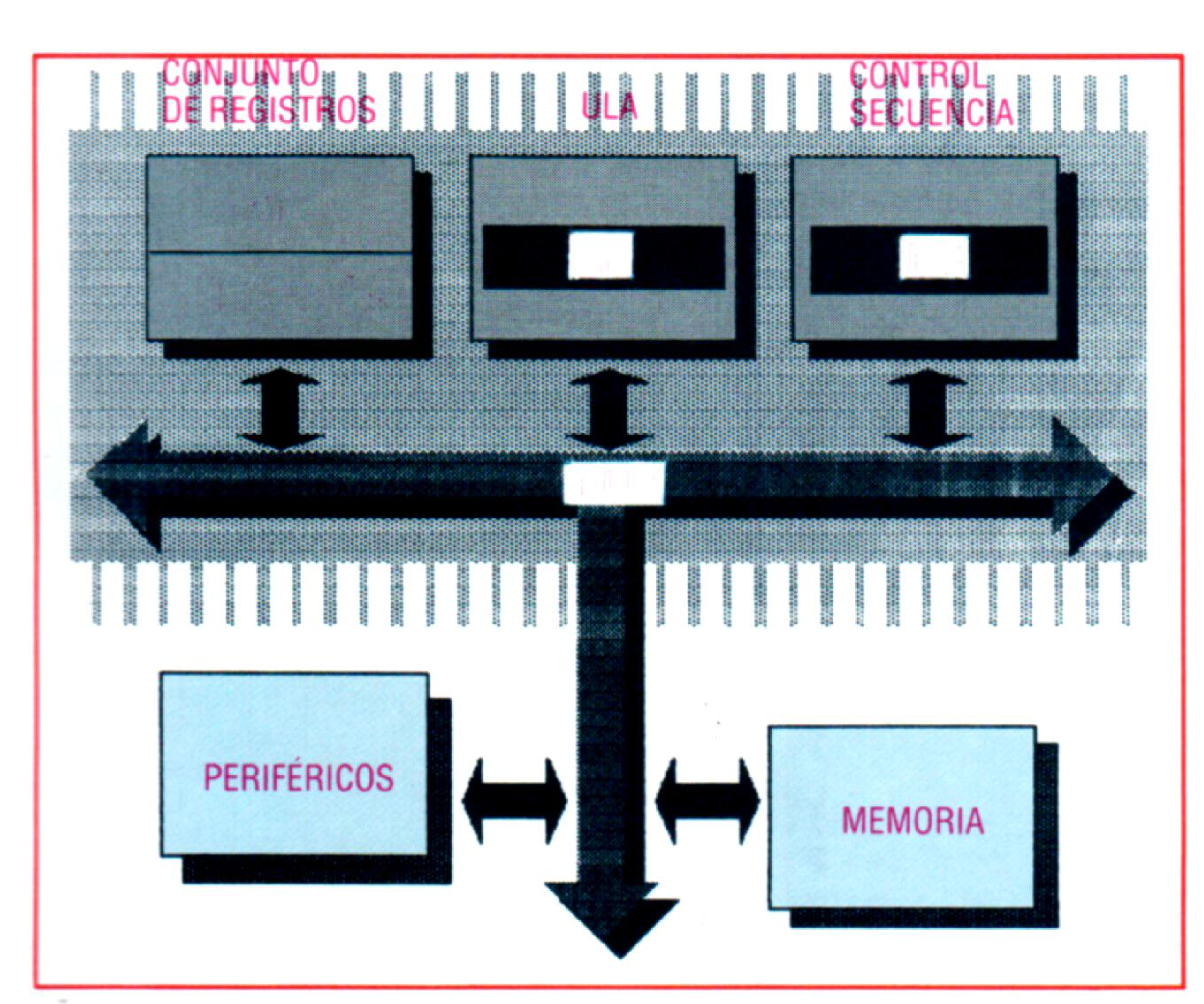
MICRO-PROCESADOR 68000
BUS
CHIP DE MEMORIA
CHIP DE INTERFACE PARA PERIFÉRICO 1
CHIP DE INTERFACE PARA PERIFÉRICO 2
VDU
IMPRESORA
ALMACENAMIENTO EN DISCO

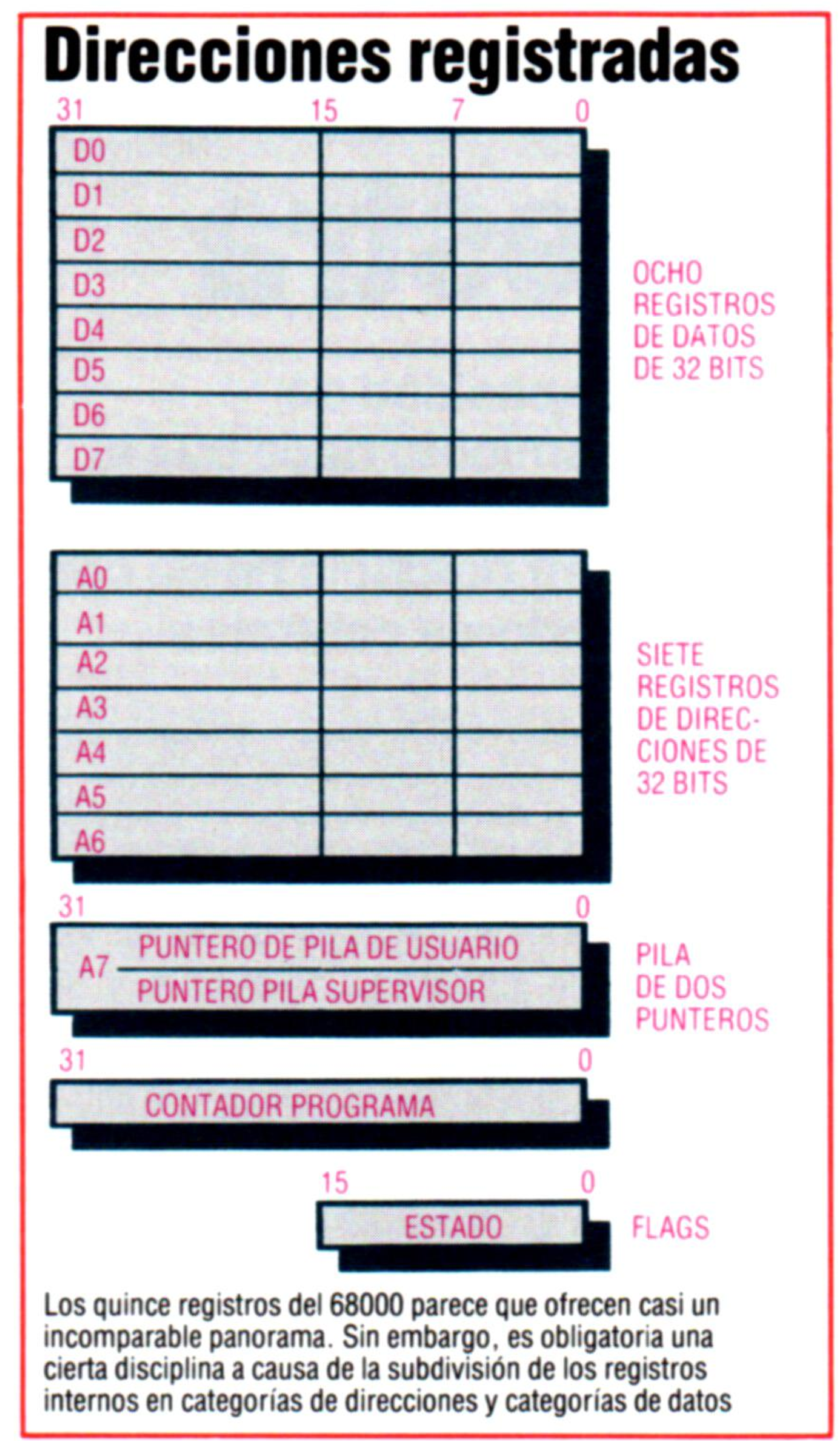
Mike Clowes

Modelo del programador
El diseño interno del 68000 puede reducirse esencialmente a tres unidades principales: los registros de datos y direcciones, la unidad aritmético-lógica y la unidad de control de secuencia. Cada unidad comunica con las restantes y con los dispositivos periféricos a través del bus de datos digital

como si el microprocesador sólo contara con un bus de comunicación central, pero debemos tener en cuenta también los componentes de nivel de registro interno.

El segundo dibujo muestra el modelo del programador donde podemos ver que la única diferencia en la arquitectura del hardware reside en que el chip del 68000 se ha escindido en tres componentes lógicos. Analicémoslos uno a uno.





Los quince registros del 68000 parece que ofrecen casi un incomparable panorama. Sin embargo, es obligatoria una cierta disciplina a causa de la subdivisión de los registros internos en categorías de direcciones y categorías de datos

Estos registros ya no son, claro está, más que una matriz de rápidas posiciones de memoria empleadas de la misma manera que cualquiera de los registros de otros ordenadores para el almacenamiento de datos o resultados intermedios. La diferencia que caracteriza al 68000 es que cuenta con ocho registros de datos y ocho registros de direcciones formados por 32 bits (dicho de otra manera, son registros de 32 bits de ancho).

No importan las razones de Motorola para escindir los registros en dos subconjuntos, lo cierto es que se han simplificado las operaciones que debe efectuar la máquina y su programación se ha hecho más fácil.

Por ejemplo, si deseamos cargar el contenido de la memoria cuya dirección se encuentra en un registro, listo para una operación aritmética, sabemos con esta arquitectura que el puntero, o la dirección, debe ser cargado en un registro de dirección y que los datos deben ser almacenados en un registro de datos. Traducido al lenguaje assembler, sería así:

```
MOVE    (A2),D4
```

donde MOVE es la instrucción de trasladar (o copiar) los datos, A2 significa la dirección de origen como contenido del registro 2 de direcciones, y D4 significa que el destino es el registro 4 de datos. Pero si la dirección de los datos de origen estuviera en D4, no podríamos usar

```
MOVE    (D4),A2
```

dado que con MOVE tanto los modos de direccionamiento usados en origen y en destino son ilegales.

Otro detalle a tener en cuenta cuando se consideran los conjuntos de registros, y en particular los registros de datos, es que los datos a los que se debe acceder pueden ser de cinco tipos:

- Bit – un solo dígito binario
- Dígito BCD (o cuarteto)
- Byte – carácter de ocho bits
- Palabra – palabra de 16 bits
- Palabra larga – palabra de 32 bits.

Trataremos los dígitos BCD y las palabras largas en capítulos venideros; de momento los consideraremos como unidades cada vez más grandes de datos.

Gran parte de las instrucciones nos permiten especificar el tipo de dato junto con la instrucción, especificándolo como un atributo de la propia instrucción. Por ejemplo, si sólo deseáramos copiar los bits del 0 al 7 desde D1 a D2 nos serviremos de:

```
MOVE.B    D1,D2
```

pero si deseáramos los 32 bits enteros, la instrucción se transformaría en:

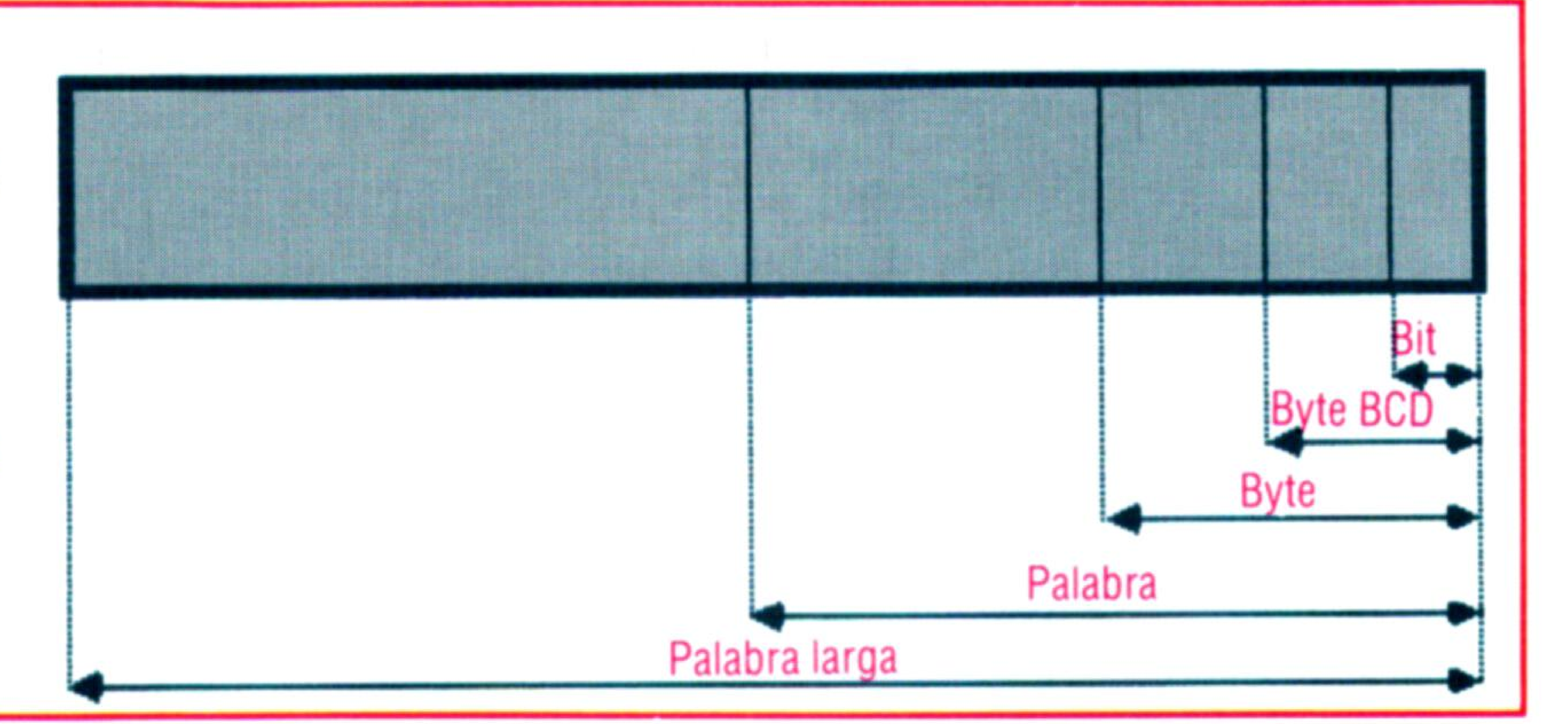
```
MOVE.L    D1,D2
```

El atributo (o código de tamaño de los datos) para una palabra de 26 bits es .W, que es el atributo por defecto cuando no se especifica ninguno.

Un último punto a notar antes de abandonar el conjunto de registros es que uno de los registros de direcciones, el A7, se emplea como puntero de pila del sistema, por lo que debemos tener cuidado de no cambiar alegremente este registro. Más tarde examinaremos las pilas.



Longitud de las palabras



El 68000 puede, mediante instrucciones ampliadas, operar con palabras de cinco longitudes distintas, procesando datos en grupos cuyo tamaño va de 1 a 32 bits. Lo cual implica las ventajas de la flexibilidad y de una mayor potencia de proceso respecto de los viejos chips de ocho bits

El siguiente elemento por considerar dentro del modelo del programador es la unidad aritmética lógica (ULA). Es la parte del micro encargada de realizar tareas aritméticas o lógicas dejando el resultado en el operando de destino.

Por ejemplo, es la ULA quien suma D1 y D2 y coloca el resultado en D2 cuando se ejecuta la siguiente instrucción:

```
ADD       D1,D2
```

Repitamos que es posible especificar el atributo longitud de los datos con la instrucción:

```
ADD.B     D1,D2
```

se limitará a sumar los bytes menos significativos.

Volviendo al diagrama del modelo del programador, existe un registro con etiqueta SR, o registro de estado, asociado a la ULA. Este registro está presente de tal modo que podemos recordar el resultado del paso de instrucción anterior. Esto se utiliza para poder efectuar una bifurcación condicional en el programa, según sea el resultado de la ejecución de la instrucción previa. Por ejemplo:

```
ADD     D1,D2     suma D1 a D2
BVS     OFLOW     comprueba desbordamiento
MOVE    D1,D3     copia D1 sobre D3
```

donde una de dos, o bien bifurcamos hasta la etiqueta OFLOW si el bit de desbordamiento está activado en SR, o bien se efectúa MOVE D1, D2 si no lo está. La instrucción BVS (*Branch if oVerflow Set*: *bifurcar si desbordamiento activado*) comprueba el desbordamiento o el bit V en el registro de estado, que puede ponerse a uno como resultado de la instrucción ADD (como ocurre en este ejemplo concreto). La condición de desbordamiento surge, como es evidente, a causa de que el resultado de una operación aritmética no se ajusta al tamaño de la palabra empleada en el operando. Si no llegáramos a detectar esta condición obtendríamos respuestas erróneas.

Una última observación sobre el registro de estado: su longitud es de 16 bits, y partes de cada byte son empleadas por el sistema de la siguiente manera:

15 — 8	7 — 0
BYTE DE SISTEMA	BYTE DEL USUARIO
	con códigos de condición

Los códigos de condición son el conjunto de bits individuales para indicar el resultado de la ejecución de la instrucción anterior: el bit V es un ejemplo de uno de estos códigos, que se estudiarán profusamente más adelante.

La sección de control de secuencias de nuestro modelo de microprocesador contiene el contador del programa, el registro que determina la dirección de la siguiente instrucción que ha de ser tomada de la memoria. Una vez tomada la instrucción, es decodificada por el control de secuencia para determinar el tipo de instrucción que la ULA ha de ejecutar y dónde están los operandos de origen y destino.

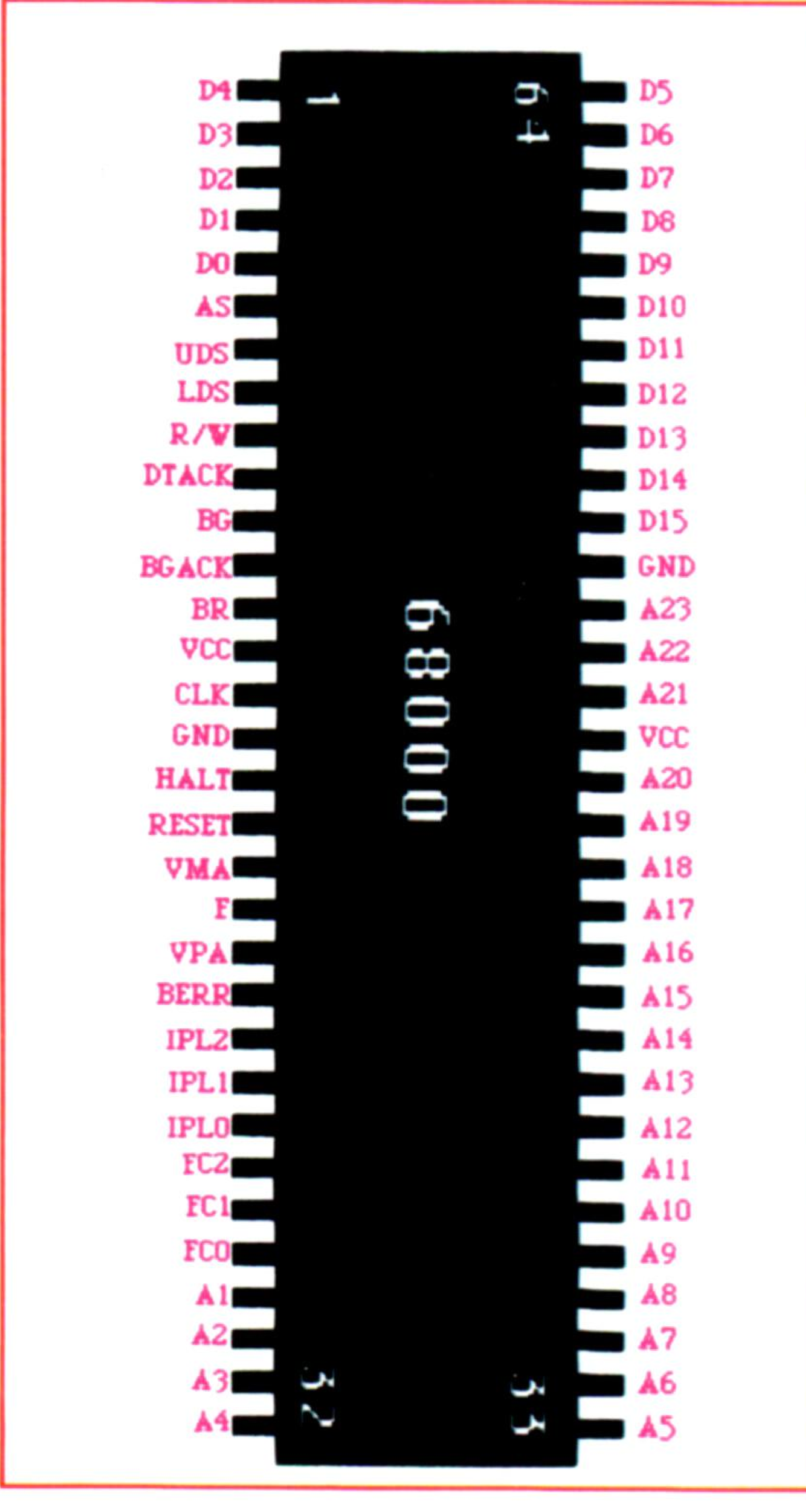
El contador del programa tiene 32 bits de largo pero las patillas a través de las cuales se conecta con el bus sólo permite el uso de 24 bits. Aun así, esto permite una gama amplísima de direcciones de memoria que llega hasta FFFFFF bytes. Cada dígito hexa corresponde a cuatro bits binarios, por lo que los 24 bits corresponden a un rango de direcciones de byte de 16 777 216. No obstante, se ha de notar que todas las instrucciones deben comenzar en una dirección impar (o frontera de palabra), lo que significa que será más comprensible pensar que hay un máximo de 8 388 608 palabras en ese rango de direcciones.

Ahora que estamos hablando de acceso a la memoria, será conveniente considerar cómo se organizan los datos en ella. Esto es necesario porque los bytes individuales son direccionables en esta máquina, y el direccionamiento de bytes en esta máquina es diferente, por ejemplo, del direccionamiento del PDP-11. El siguiente esquema ilustra el direccionamiento de la memoria del 68000 mostrando cómo la dirección impar proporciona el byte más significativo de la palabra.

	15 — 8	7 — 0
Pal. n	byte n	byte n+1
Pal. n+2	byte n+2	byte n+3

Algunas instrucciones cambiarán directamente el contador del programa a fin de provocar una bifurcación, bien sea incondicional como BRA BACKHERE (donde BRA quiere decir *branch always* [bifurcar siempre] a una dirección representada simbólicamente por la etiqueta BACKHERE), bien condicional como en el ejemplo anterior, BVS OFLOW, en el que la bifurcación depende de si el bit V está a uno en la palabra del PS. En ambos casos, cuando ocurre una bifurcación o cambio en el flujo secuencial de ejecución normal, el contador del programa se modifica para apuntar a la siguiente instrucción a ejecutar.

Patillas a granel
Aquí están las patillas de comunicación del 68000. Obsérvense las 24 líneas de direcciones y las 15 de datos. A diferencia de su pariente más próximo, el 68008 (que posee un bus de direcciones de sólo 8 bits), el 68000 puede direccionar en modo directo dentro del ámbito del 000000 al FFFFFF hexa, es decir, del 0 al 16 177 216. Su primo mayor, el 68020, tiene un bus de direcciones de 32 bits completo. El 68000 es un perfecto ejemplo de microprocesador moderno: más de 100 000 circuitos lógicos se combinan para obtener la potencia de proceso de un miniordenador en un único chip



El conjunto de instrucciones

Los fabricantes de ordenadores suelen jactarse del número de instrucciones independientes que pueden soportar sus máquinas. Esto puede confundir a más de uno puesto que a menudo es necesaria una instrucción distinta para cada tipo de datos de operando o modo de direccionamiento. Esto es del todo verdadero en los antiguos ordenadores de ocho bits en los que, por ejemplo con el 8085, hay disponibles 63 instrucciones distintas de MOV, y cada una se sirve de los registros en una combinación diferente.

Más importante que el número absoluto de códigos de instrucción es la gama de instrucciones, sus objetos de datos y la flexibilidad de los modos de direccionamiento que pueden ser empleados con las instrucciones.

Motorola ha previsto un conjunto de instrucciones poderosísimo en lo que respecta a objetos de datos, que nos permitirá direccionar bytes, palabras y palabras largas con la mayoría de las instrucciones. Podemos realizar la multiplicación, división binaria y la aritmética BCD, así como la gama habitual de instrucciones de operaciones lógicas, control de programa y copia de datos.

No obstante, aun cuando se dispone de una gama muy útil de modos de direccionamiento, no podemos usarlos sin estudiar antes la instrucción. Por ejemplo, si el operando de destino para la instrucción MOVE es un registro de dirección, debemos emplear la instrucción MOVEA o bien LEA. En otras palabras, no podemos escribir MOVE D3,A6 pero sí que podemos MOVEA D3,A6. La razón es que al proporcionar un conjunto de instrucciones tan poderoso con respecto a los objetos de datos, se ha sacrificado una pequeña parte de la capacidad de direccionamiento.

En su totalidad el conjunto de instrucciones es muy potente en comparación con los micros de ocho bits (incluidos algunos miniordenadores y ordenadores centrales) con un excelente abanico de objetos de datos, pero con una flexibilidad decepcionante en los modos de direccionamiento utilizables con las instrucciones. En algo se compensa esto gracias a la densidad de algunas instrucciones (conocidas como instrucciones *rápidas*), donde la instrucción entera se codifica en una palabra de máquina.

Por ejemplo, para establecer a 25 un registro de datos escribiríamos:

```
MOVEQ#25,D3
```

que llevará 25 a D3 y la instrucción entera sólo ocupará una palabra, incluyendo el dato constante 25.

En el próximo capítulo estudiaremos con mayor detalle los modos de direccionamiento disponibles y cómo los podemos usar para acceder a los datos. Incluiremos algunos ejemplos de programas.

Por delante de su época

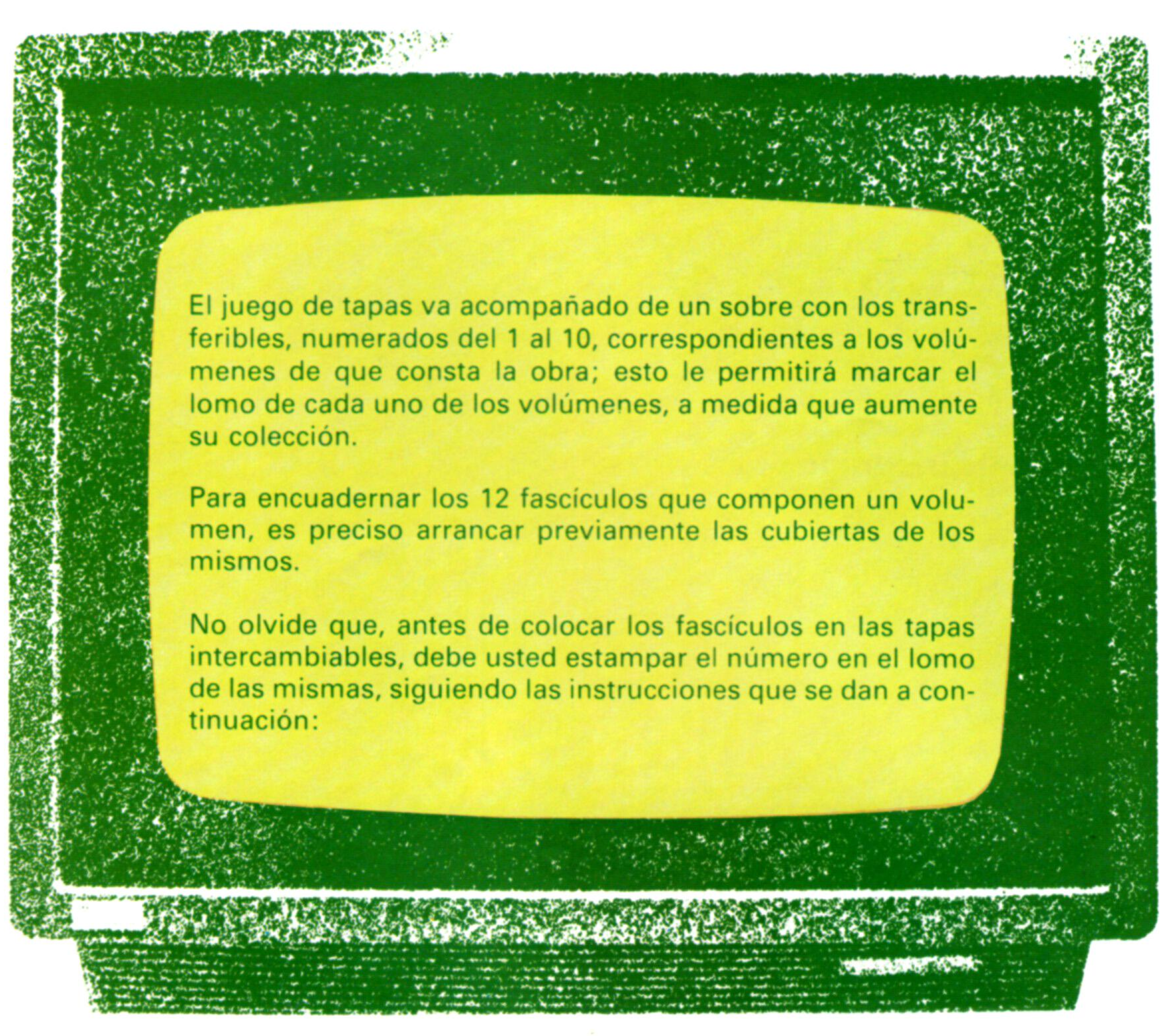
El Motorola 6809 llegó demasiado tarde como para causar impacto en el *boom* de los micros domésticos a finales de los años setenta y comienzos de los ochenta. El 68000 de 16 bits fue lanzado cuando muchos fabricantes estaban diseñando una nueva generación de máquinas, por lo que se ha incorporado a ordenadores como el QL de Sinclair, el Macintosh de Apple y el ST de Atari.

Desde el punto de vista del programador, es una delicia trabajar con este chip. El procesador tiene 17 registros de 32 bits, un bus de datos de 16 bits y un bus de direcciones de 24 bits, además del juego de instrucciones.

No obstante, y a pesar del impresionante acabado de muchos de los ordenadores que hasta hoy han incorporado el 68000, existen todavía problemas por resolver antes de que se logre utilizar su completo potencial. El problema básico hasta el momento ha sido el que el 68000 es tan adelantado en su tecnología respecto a otros chips que los fabricantes de ordenadores lo están empleando muy por debajo de sus posibilidades. Un ejemplo extremo es el QL de Sinclair. Este ordenador está basado en la versión 68008 del chip, que sólo tiene un bus de datos de ocho bits. Con lo cual, el QL no es, para muchas aplicaciones, mucho más rápido que sus predecesores de ocho bits. Sin embargo, el futuro del procesador está en el chip 68010, que presenta un bus de direcciones de 16 bits pero que ofrece el soporte de una memoria virtual. Esto permite que parte de la memoria resida en un dispositivo de almacenamiento de fondo más barato, permitiendo así el empleo de mucha más memoria de la que se podría disponer de otro modo. Otro chip de Motorola —el 68020— ha sido saludado como un ordenador de 32 bits en un chip que ofrece ¡el 97 % de la potencia de un ordenador central!



Con el fascículo anterior se han puesto a la venta las tapas correspondientes al noveno volumen.



El juego de tapas va acompañado de un sobre con los transferibles, numerados del 1 al 10, correspondientes a los volúmenes de que consta la obra; esto le permitirá marcar el lomo de cada uno de los volúmenes, a medida que aumente su colección.

Para encuadernar los 12 fascículos que componen un volumen, es preciso arrancar previamente las cubiertas de los mismos.

No olvide que, antes de colocar los fascículos en las tapas intercambiables, debe usted estampar el número en el lomo de las mismas, siguiendo las instrucciones que se dan a continuación:



1 Desprenda la hojita de protección y aplique el transferible en el lomo de la cubierta, haciendo coincidir los ángulos de referencia con los del recuadro del lomo.

2 Con un bolígrafo o un objeto de punta roma, repase varias veces el número, presionando como si quisiera borrarlo por completo.

3 Retire con cuidado y comprobará que el número ya está impreso en la cubierta. Cúbralo con la hojita de protección y repita la operación anterior con un objeto liso y redondeado, a fin de asegurar una perfecta y total adherencia.

Editorial Delta

Cada sobre de transferibles contiene una serie completa de números, del 1 al 10, para fijar a los lomos de los volúmenes. Ya que en cada volumen sólo aplicará el número correspondiente, puede utilizar los restantes para hacer una prueba preliminar.

Ya están a su disposición, en todos los quioscos y librerías, las tapas intercambiables para encuadernar 12 fascículos de

mi COMPUTER

Cada juego de tapas va acompañado de una colección de transferibles, para que usted mismo pueda colocar en cada lomo el número de tomo que corresponda





